Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FINAL DE CARRERA

TÍTULO DEL TFC: Diseño e implementación de un sistema de adquisición y actuación inalámbrico para vehículos aéreos no tripulados.

TITULACIÓN: Ingeniería Técnica Aeronáutica, especialidad Aeronavegación

AUTOR: Jorge Polo Alonso

DIRECTOR: Ramón Casanella Alonso

FECHA: 3 de junio de 2013

Título: Diseño e implementación de un sistema de adquisición y actuación inalámbrico para vehículos aéreos no tripulados.

Autor: Jorge Polo Alonso

Director: Ramón Casanella Alonso

Fecha: 3 de junio de 2013

Resumen

En la actualidad está ocurriendo una revolución en el mundo de la aeronáutica. Desde hace unos años se ha producido una nueva línea de tendencia a utilizar vehículos aéreos no tripulados, principalmente para el uso militar, pero poco a poco este tipo de vehículos se están destinando a una gran variedad de misiones de uso civil, desde vigilancia aérea hasta proyectos basados en detección y extinción de incendios.

Visto el potencial de estas aeronaves y su clara evolución de futuro, se ha definido este proyecto con el objetivo de diseñar e implementar un sistema de adquisición y actuación inalámbrico para este tipo de vehículos aéreos, concretamente para una aeronave de pequeñas dimensiones y de coste reducido, que pueda ser destinada a la enseñanza.

Además del diseño del sistema, la presente memoria pretende ser un documento de carácter didáctico, orientado a ofrecer una guía para el estudio y el diseño de un sistema de adquisición y actuación. Este proyecto quiere ofrecer una línea de desarrollo que va desde la comparativa y selección de los diferentes componentes hasta la creación de herramientas propias, como puede ser por ejemplo, el diseño de un sencillo protocolo de comunicación adaptado a las necesidades del sistema.

Finalmente, el presente proyecto pretende ser un sistema adaptable a las necesidades futuras o ajenas, estableciendo una pequeña base en diferentes campos relacionados con el diseño de sistemas destinados a ser embarcados en aeronaves no tripuladas de reducido tamaño, como por ejemplo: la programación, las comunicaciones, el diseño del hardware, el acondicionamiento de sensores, etc.

Title: Analysis and design of data acquisition systems and wireless control for unmanned aerial vehicles.

Author: Jorge Polo Alonso

Director: Ramón Casanella Alonso

Date: June, 3rd 2013

Overview

Nowadays, a revolution in aeronautics is taking place. Since some years ago up to this moment, a new trend has been brought forth and it's increasingly growing: using unmanned aerial vehicles, mainly for military purposes, but also there is an increasing trend to use them for a wide array of civilian missions (from aerial vigilance to projects geared at detecting and extinguishing fires).

Given the potential of these aerial vehicles and its clear future possibilities, this project has been carried out with the aim of designing and implementing a wireless system of acquisition and actuation for these sorts of aerial vehicles. Mainly, it has been thought for a small-sized, cost-effective aerial vehicle, which could be used for teaching.

Apart from the system design, this report is also intended to be an educational document, geared at offering guidelines for the study and design of acquisition and actuation systems. Thus, this report aims to offer a development line that encompasses from the comparison and selection of the different components to the creation of new tools, e.g. the design of an easy-to-use communications protocol adapted to the system needs.

Finally, this report intends also to offer an adaptable system that could suit future needs, by setting up contributions in different fields related to the design of systems destined to be embarked in small-sized unmanned aerial vehicles, such as for example: communications, hardware design, sensors conditioning, etc.

A mi familia, por su infinita paciencia.

ÍNDICE

INTRODUCCIÓN Y OBJETIVOS

Este trabajo de final de carrera se presenta con el objetivo de crear un sistema de adquisición y actuación; orientado al entorno de los vehículos aéreos no tripulados y concretamente a los MAVs, siglas de *Micro Air Vehicles*.

Un MAV es una aeronave no tripulada de reducidas dimensiones que viene definida por un peso no superior a 5 Kg y un alcance inferior a los 10 Km [1]. Este tipo de aeronave se engloba dentro de los vehículos aéreos no tripulados o UAVs, *Unmanned Air Vehicles* o popularmente conocidos como *drones*, aunque este último término, normalmente se aplica exclusivamente al campo militar o cuando el vehículo vuela de forma autónoma, sin necesidad de control de un piloto.

El uso de un MAV es muy diverso, en este sentido podemos encontrar desde aplicaciones militares hasta proyectos de investigación basados en inteligencia artificial. En la figura 0.1 podemos ver dos tipos de MAV's. El primero de ellos, imagen de la izquierda, corresponde al dispositivo Black Hornet Nano, dispositivo de reconocimiento orientado al apoyo de la infantería. El segundo, imagen de la derecha, corresponde a un MAV desarrollado por el Instituto de Tecnología de Massachusetts (MIT, en sus siglas en inglés) y tiene como objetivo el estudio y el desarrollo de aplicaciones basadas en visión por computador.

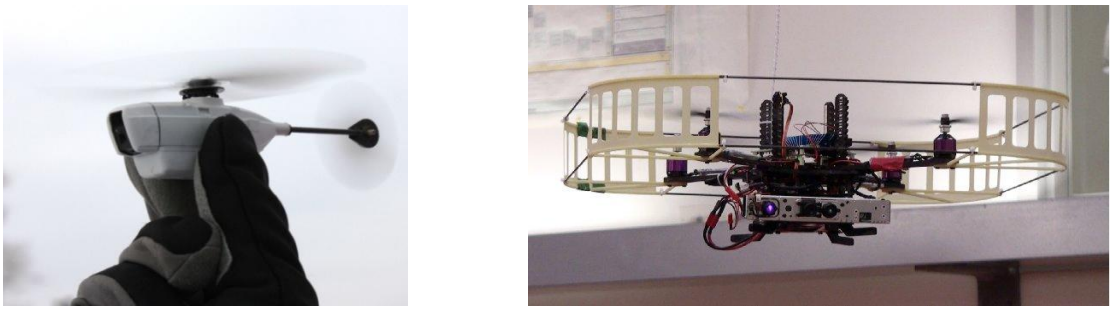


Fig. 0.1 Ejemplos de MAV's. [2]

En este trabajo se pretende potenciar una faceta muy concreta de los MAV's, su uso como herramienta docente. En concreto este trabajo se basa en desarrollar un sistema capaz de adquirir datos de sensores y controlar actuadores, pero se espera que pueda ser una base para desarrollar otro tipo de aplicaciones por ejemplo, sistemas de guiado o incluso sistemas basados en inteligencia artificial.

Los requisitos de este proyecto se expondrán seguidamente, no obstante, se quiere dejar claro al lector, que este proyecto se enmarca dentro de una ingeniería técnica aeronáutica, y su fin último es usar el sistema desarrollado

como una base para ser implementado dentro de una pequeña aeronave destinada a la docencia, aunque el diseño y construcción de esta plataforma no sea abordado en este proyecto.

En definitiva este proyecto pretende dar soporte a cualquier estudiante de ingeniería, aportando las herramientas necesarias para desarrollar una base en el estudio de sistemas de adquisición y control, con especial énfasis en el diseño de la unidad de control, para que después cada uno pueda adaptar lo aprendido a sus necesidades.

Como se ha definido anteriormente, una de las partes principales del sistema tiene como objetivo principal el análisis y el diseño de un sistema de adquisición de datos o SAD. Más conocidos con sus siglas en inglés DAS, *Data Adquisition System*, este tipo de sistemas llevan a cabo una labor indispensable en la gran mayoría de campos de la ingeniería, son los encargados de obtener y representar la información que proviene de los sensores. Los sensores son los encargados de extraer información de una magnitud física en concreto cómo puede ser: la velocidad, el rumbo, la temperatura, etc.

El propósito de este proyecto de final de carrera es el de analizar e implementar un sistema de adquisición que además también sea capaz de controlar actuadores, concretamente motores. El sistema tiene que poder trabajar de forma inalámbrica, utilizando una estación base para la comunicación con el ordenador y una estación remota capaz de adquirir datos de diferentes sensores y controlar la actuación de la aeronave.

Siempre que se empieza el desarrollo de un sistema, se tiene que tener en cuenta una serie de criterios, llamados parámetros de diseño o lo que es lo mismo, establecer los requisitos que el sistema debe cumplir. Para este proyecto se han establecido una serie de requisitos previos. El primero de estos requisitos, es que el sistema debe permitir familiarizarse con modelos básicos de acondicionamiento y calibración de sensores. El segundo criterio es el precio, el sistema debe ser lo más económico posible. El tercero es que se diseñe un sistema de forma modular o lo que es lo mismo, que el sistema pueda ser divido conceptualmente en partes individuales. El cuarto es que las herramientas se diseñen a través de la plataforma que nos ofrece la comunidad Arduino. El quinto es que los componentes de sistema deben ser de fácil adquisición; por lo que se ha usado una única fuente de distribución de componentes, que ha sido la empresa Farnell® o cualquiera de sus distribuidores asociados. El sexto y último criterio, es que el sistema se englobe en un entorno destinado a ser embarcado en vehículos aéreos no tripulados, por lo que un criterio básico es el espacio y el peso que ocuparán los componentes del sistema.

Con referencia al sexto y último requisito, es importante concretar que los componentes utilizados en este proyecto (condensadores, resistencias, chips, etc...) no serán con encapsulado SMD o similares, esto aunque parezca una contradicción a la hora de conseguir el mínimo peso y espacio posible, se hace porque este proyecto es sobre todo un proyecto con un fin didáctico basado en

la experimentación. Se ha considerado oportuno que todos los elementos del sistema puedan ser probados y testados en placas de pruebas o protoboards.

Establecidos ya los requisitos del sistema, podemos ver la estructura de la que se compone este proyecto. Este proyecto se divide imitando la estructura de cualquier sistema SAD añadiendo la parte de actuación, en la figura 0.2 podemos ver una representación básica de cómo está estructurado nuestro sistema.

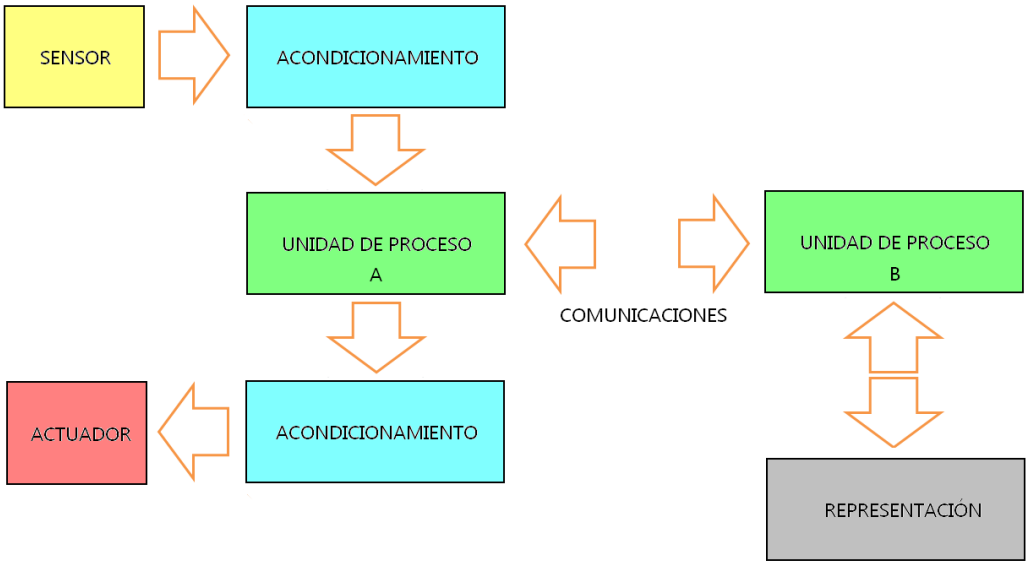


Fig. 0.2 Estructura del sistema de adquisición y actuación del proyecto

Basándonos en esta arquitectura, el proyecto se divide en tres grandes temas o capítulos. El primero de los capítulos corresponde a la unidad de procesamiento, este elemento se ha constituido el primer elemento del proyecto porque influye profundamente en los demás. El segundo capítulo corresponde a la parte de los sensores, actuadores y su acondicionamiento, en esta parte se verá los sensores elegidos para el SAD y los criterios que se han seguido para su elección. En el tercer capítulo se hablará de la representación de la información y cómo se ha desarrollado la interfaz gráfica mediante el programa LabVIEW®.

Además de los apartados ya descritos, que corresponden a la estructura básica, se verán otros puntos como pueden ser, un apartado destinado a las comunicaciones del sistema, una parte destinada a la programación o una vez diseñado el sistema, su implementación sobre una placa de circuito impreso o PCB, siglas de *Printed Circuit Board.*

CAPÍTULO 1. UNIDAD DE PROCESAMIENTO

La unidad de procesamiento es la parte central del sistema, es la encargada de recibir la señal y codificarla para que pueda ser entendida y procesada. Como analogía podemos decir que la unidad de procesamiento sería el cerebro del sistema.

El elemento más importante dentro de la unidad de procesamiento es el CAD, Conversor Analógico Digital o en sus siglas en inglés ADC, *Analog Digital Converter.* El ADC será el elemento que nos permitirá transformar la señal analógica que procede del sensor a una señal digital que pueda ser procesada.

1.1 Eligiendo la plataforma

La unidad de procesamiento es una parte crítica en un sistema SAD porque nos determina el acondicionamiento de los sensores. Según el ADC elegido y la tecnología utilizada en el microcontrolador nos veremos limitados a unos determinados parámetros de diseño (niveles de resolución, niveles de voltaje de entrada, velocidad de procesamiento, etc.). En la actualidad, muchos microprocesadores llevan integrado su propio ADC y gracias a esto, podemos concentrar toda nuestra unidad de procesamiento en un único chip. Esta es la solución elegida para este proyecto.

Aunque el primer paso a realizar sería elegir una plataforma de desarrollo, basada en una u otra familia de chips (PIC®, Atmel®, Texas Instruments®, etc.), en este proyecto se ha elegido la plataforma Arduino ya que esta ha sido facilitada por la universidad y se ha establecido como requisito de diseño.

Una plataforma de desarrollo como Arduino, nos proporciona además del microcontrolador otros elementos, por ejemplo, un método de comunicación entre el microprocesador y el ordenador basado en comunicación vía USB, lo que nos ayudará en la comunicación y en la programación. Arduino además nos proporciona su propio entorno de programación, con un lenguaje basado en C, que nos ayudará a cargar programas en el microcontrolador para que puedan ser ejecutados.

La plataforma facilitada por la universidad ha sido “Arduino Uno Rev. 3”. Esta plataforma se basa en un microcontrolador ATMEL328P-PU, cuyas características más importantes se representan en la tabla 1.1.

Tabla 1.1. Especificaciones básicas de la plataforma Arduino UNO Rev. 3 [3]

Microcontrolador	ATmega328P-PU	Memoria EEPROM	1 KB
CPU	8 bits	Pines Digitales	14
Frecuencia Reloj	16 MHz	Pines PWM	6
Voltaje Operación	5V	Pines Analógicos	6
Memoria FLASH	32 KB	Resolución ADC	10 bits
Memoria SRAM	2 KB	Intensidad salida de Pines	40 mA

1.2 Introducción a Arduino

Arduino es una plataforma de hardware y software libre basada en los de chips de la marca Atmel®. En los últimos años esta plataforma se ha ganado muchos seguidores debido a un lenguaje de programación de alto nivel basado en C que permite la programación del microcontrolador de una manera muy sencilla.

Arduino permite a un usuario con unos conocimientos básicos en programación programar el microcontrolador integrado en la plataforma y así realizar una multitud de proyectos que van desde encender un led a introducirse en el mundo de la robótica o incluso realizar proyectos más avanzados basados en comunicaciones inalámbricas o Ethernet.

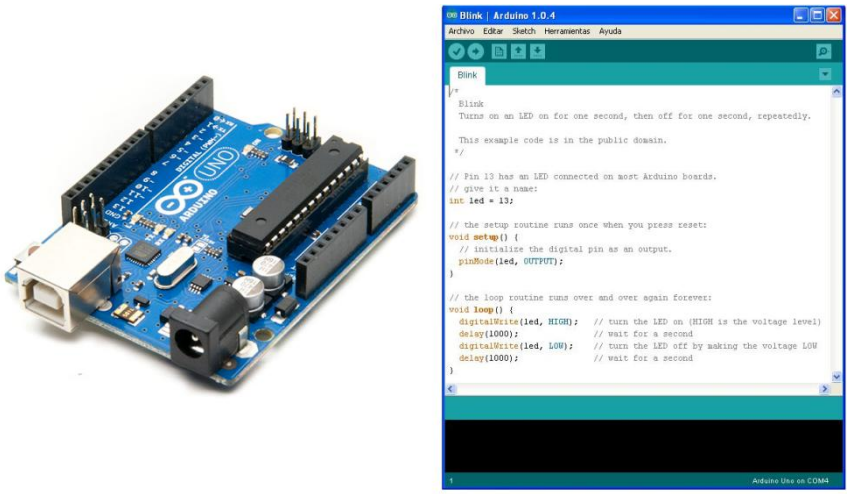


Fig. 1.1 Plataforma Arduino [4]

En la figura 1.1 podemos ver una imagen de la plataforma Arduino y una ventana con un programa listo para ser compilado y ejecutado en el microcontrolador.

Como vemos Arduino nos proporciona todas las herramientas necesarias para programar el microcontrolador, desde los drivers que necesita el ordenador hasta el entorno de programación necesario para empezar a realizar nuestros proyectos. Además de todo esto, Arduino tiene dos grandes ventajas, la primera es que existe una comunidad muy numerosa que da soporte a esta plataforma, y segundo existen módulos de ampliación que permiten incorporar a la placa utilidades adicionales (comunicación WIFI, control de motores, etc.).

Para más detalle sobre esta plataforma se puede consultar la página web de la comunidad: www.arduino.cc.

1.3 Arquitectura de Arduino

Para conocer mejor las posibilidades que nos ofrece Arduino debemos conocer un poco más su arquitectura, esto significa entender las entradas y salidas que podemos utilizar para realizar nuestros proyectos.

Una de las ventajas que nos ofrece Arduino es que todos los pines que podemos utilizar están numerados haciendo así más fácil interactuar con la plataforma y facilitándonos así la forma de programar.

Primero de todo tenemos dos grandes grupos de pines, los pines digitales y los pines analógicos.

Los pines analógicos son claves en nuestro proyecto ya que son los que están conectados al conversor ADC. En Arduino Uno tenemos 6 pines analógicos numerados desde el A0 hasta el A6. Cada uno de estos pines es capaz de digitalizar una señal de entrada entre los 0V y VCC, convirtiendo el voltaje de entrada en un valor numérico que va desde el cero hasta el 1023, este rango de valores corresponde a la resolución del ADC que es de 10bits.

Estos pines nos serán de gran utilidad para transformar la señal procedente del sensor a una información muy fácil de procesar.

El segundo grupo de pines son los digitales, estos pines nos sirven tanto como de entrada como de salida, esto quiere decir que mediante la programación podemos poner el pin con una tensión de 0V que corresponde al nivel bajo o LOW o poner una tensión de 5V para obtener un nivel alto o HIGH. Como vemos con cualquiera de estos pines podemos realizar infinidad de circuitos basados en sistemas binarios, desde usar los pines como interruptores a enviar señales digitales enviando pulsos de unos y ceros. Los pines digitales en Arduino Uno están numerados del 0 al 13.

Una característica muy útil, sobretodo, para el control de motores o actuadores son los pines PWM, estos pines corresponde a los pines digitales 3, 5, 9, 10 y 11 y tienen la ventaja que pueden ser modulados en frecuencia, de momento nos basta con saber que con estos pines seremos capaces de sacar voltajes intermedios entre 0V y 5V. Por ejemplo con estos pines podríamos graduar la velocidad de un pequeño motor de corriente continua. En la figura 1.2 vemos una imagen con la distribución de los pines comentados en la placa.

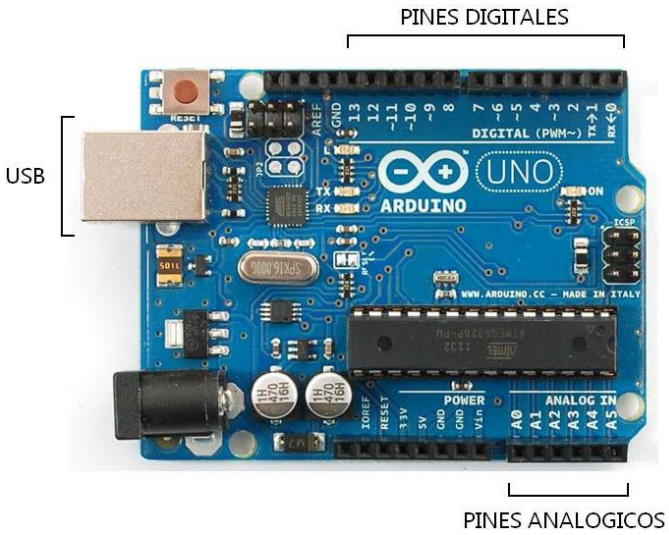


Fig. 1.2 Pines de Arduino UNO [5]

1.4 Programando con Arduino, una muy breve introducción.

Una vez vista brevemente la arquitectura, y que somos capaces de situar los pines en la placa, veamos un poco como se programa con Arduino.

Como ya se ha comentado a lo largo del capítulo, Arduino está basado en un microcontrolador Atmel®, la ventaja de Arduino es que nos permite programar las salidas que hemos comentado anteriormente con una gran facilidad, aunque en este proyecto no se entrará en detalle a niveles de programación, se verán unos pequeños ejemplos que hagan entender a una persona sin conocimientos previos, lo sencillo que es programar un microcontrolador mediante esta plataforma.

Lo primero que se debe hacer es conectar Arduino mediante un USB a nuestro ordenador e instalar los drivers que nos proporciona la comunidad. A partir de ahí y a través del programa que también se proporciona, se puede empezar a programar el microcontrolador. Existen infinidad de páginas donde encontraremos programas para realizar con Arduino, pero para empezar, Arduino te proporciona ejemplos básicos para realizar pruebas con tu plataforma, entre los ejemplos más básicos esta Blink, que nos permite encender el led integrado en la placa, este ejemplo se incluye en la figura 1.3.

Todas las instrucciones que se pueden usar en arduino se pueden consultar en la siguiente dirección: http://arduino.cc/es/Reference/Extended

1.4.1 Estructura básica de un programa en Arduino

En la gran mayoría de programas de Arduino tendremos la misma estructura de programa, básicamente tendremos tres partes: una parte donde se declaran las variables y/o las librerías, otra parte que corresponde al procedimiento *setup*, donde se inicializan los pines digitales a utilizar, y la otra que corresponde al programa principal, el procedimiento *loop*, donde se ejecutará un programa en un bucle infinito.

```
int led = 13;

/* Asignamos a una variable "led" el pin que
   queremos usar como salida. */

void setup() {
  pinMode(led, OUTPUT);      // Asignamos el pin como salida
}

void loop() {
  digitalWrite(led, HIGH);   // Salida del Pin a VCC     -> LED ON
  delay(1000);               // Tiempo de espera 1 segundo
  digitalWrite(led, LOW);    // Salida del Pin a 0V      -> LED OFF
  delay(1000);               // Tiempo de espera 1 segundo
}
```

Fig. 1.3 Programa básico en Arduino. [6]

1.5 Bootloader Arduino

La primera pregunta que se nos plantea es, ¿qué es un bootloader? Un bootloader, gestor de arranque en castellano, es un pequeño programa que contiene el chip que permite programar el chip utilizando las herramientas que proporciona Arduino, por ejemplo sin este bootloader, no sería posible utilizar el lenguaje de programación propio de Arduino.

La ventaja más importante es que mediante este bootloader podemos convertir un chip "vacío" en un chip fácilmente programable a nuestras necesidades, sin este gestor de arranque el chip debería ser programado con otros lenguajes y conceptos de programación un poco más avanzados.

A la pregunta, si ya tengo una placa Arduino UNO, ¿por qué necesito programar un chip Atmel vacío? Pues muy sencillo, por precio. Una placa Arduino parecida a la usada en este proyecto vale mínimo 15 euros [7], un chip ATmega328P-PU vale alrededor de 3 euros [8], además tener solo el chip permite una versatilidad mayor, por ejemplo permite ser embarcado en dispositivos cuyo parámetro crítico es el peso, como es nuestro caso.

Para introducir el bootloader en un chip vacío tenemos dos opciones: usar el Arduino como ISP, In System Programming, o usar un hardware conocido como programador.

La comunidad Arduino proporciona los bootloader necesarios para realizar la conversión de un chip vacío para poder ser utilizado mediante las herramientas de programación de Arduino, un bootloader no deja de ser un archivo en formato hexadecimal que se carga en las posiciones inferiores de la memoria del chip.

Una vez introducido el bootloader tendremos un chip que podemos embarcar en un dispositivo móvil, esto nos puede ser muy útil para realizar pruebas, o para aprender técnicas de comunicación inalámbricas. Otra alternativa sería usar plataforma Arduino de tamaño reducido, como por ejemplo Arduino Mini o Arduino Micro, pero que se alejan de este proyecto, por no tener tanta flexibilidad, y por ser alternativas más caras.

1.6 Extendiendo Arduino a otros microcontroladores

Como ya hemos comentado la flexibilidad que nos proporciona la comunidad Arduino tiene muchas ventajas, entre ellas existe una particularmente interesante, podemos exportar aplicaciones a otros microcontroladores más apropiados a nuestras necesidades.

Atmel® tiene una gran gama de microcontroladores de 8 bits, entre ellos y dada su pequeña dimensión podemos destacar los microcontroladores de la familia tinyAVR®. La comunidad Arduino y especialmente comunidades como GitHub [9], High-Low Tech [10] y Arduino-Tiny [11] han desarrollado librerías y bootloaders, para exportar Arduino a este tipo de controladores, además de

ofrecer una enorme cantidad de documentación excelente para el desarrollo de estos dispositivos.

En la imagen 1.4 vemos dos tipos de microcontroladores de la familia tinyAVR® que se han barajado para dotar al proyecto de un sistema remoto, el Attiny84 y el Attiny85. [12]

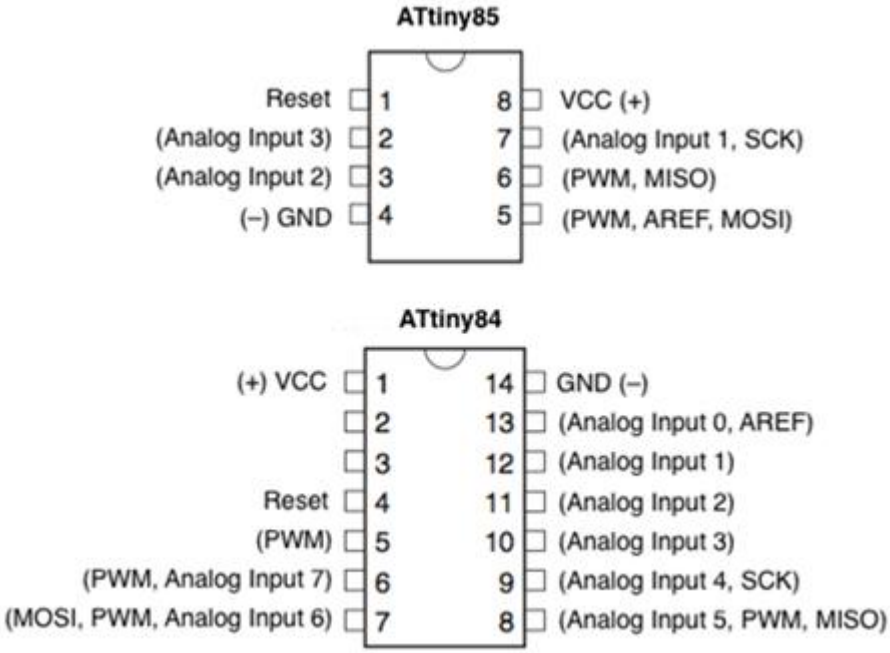


Fig. 1.4 Microcontroladores ATtiny84 y ATtiny85 [13]

Los microcontroladores ATtiny mencionados se pueden programar en Arduino aunque tienen una serie de limitaciones de código, y sólo permiten determinados comandos de Arduino, aunque para este proyecto serán más que suficientes. Para programarlos necesitaremos nuevas librerías, que también son proporcionadas libremente. En este proyecto se ha escogido la librería que proporciona Arduino-Tiny. La librería se puede descargar en la siguiente dirección: http://arduino-tiny.googlecode.com/files/arduino-tiny-0100-0015.zip.

Como opción alternativa el lector puede descargar: https://github.com/damellis/attiny/. Esta segunda librería la proporciona la comunidad High-Low Tech, que es un grupo de desarrollo del MIT. La opción de decantarse por la primera es que la primera contiene los archivos de bootloader para nuestros microcontroladores.

El tutorial para realizar la programación de estos dispositivos y como instalar las librerías se puede encontrar en la página: http://hlt.media.mit.edu/?p=1695. No se ha considerado necesario traducir el Tutorial, porque es bastante sencillo de entender, aunque sí que en este proyecto se aclararán algunos conceptos como la programación ISP, en siguientes apartados. Ahora ya tenemos dos poderosas herramientas, Arduino UNO y un microcontrolador ATtiny.

1.6.1 Eligiendo microcontrolador tinyAVR®.

Ahora mismo tenemos tres microcontroladores para realizar el SAD, la intención inicial es usar Arduino UNO como estación base y un ATtiny como estación remota, y establecer una comunicación inalámbrica entre ellos. La pregunta en este punto es ¿Qué elegimos un ATtiny85 o un ATtiny84? Básicamente la gran diferencia entre ellos, ya que tienen una arquitectura similar, es el número de pines. La tabla 1.2 nos muestra una comparativa entre los dos microcontroladores, y vemos que salvo el número de pines son idénticas.

Tabla 1.2. Comparativa ATiny84 vs ATtiny85 [14]

Microcontrolador	ATtiny85	Memoria EEPROM	512 bytes
CPU	8 bits	Pines Digitales	6
Frecuencia Reloj	20 MHz***	Pines PWM	2
Voltaje Operación	2.7 – 5.5V**	Pines Analógicos	3
Memoria FLASH	8 KB	Resolución ADC	10
Memoria SRAM	0,5 Kbytes	Precio* REF: ATTINY85-20PU	1,53 €
Microcontrolador	ATtiny84	Memoria EEPROM	512 bytes
CPU	8 bits	Pines Digitales	12
Frecuencia Reloj	20 MHz***	Pines PWM	4
Voltaje Operación	2.7 – 5.5V**	Pines Analógicos	8
Memoria FLASH	8 KB	Resolución ADC	10 bits
Memoria SRAM	0,5 Kbytes	Precio* REF: ATTINY84-20PU	3,01 €

*** Máxima frecuencia con cristal u oscilador externo.
** Valor que depende de la frecuencia de operación, de 4.5 – 5.5V a 16 MHz
* Precio y referencia de http://es.farnell.com , encapsulado DIP, 15/04/2013

Como vemos que los dos microcontroladores, son iguales, se programan igual, ahora mismo la elección es trivial y dependerá del número de pines a utilizar, pero llegados a este punto empezaremos utilizando el ATtiny84.

La explicación es sencilla, este tipo de microcontroladores, pueden trabajar a diferentes frecuencias (1, 8, 16 o 20 MHz), como facilidad en el uso entre Arduino UNO y el ATtiny se ha elegido trabajar a la misma frecuencia de operación, 16 MHz. Para adaptar el ATiny a esta frecuencia de trabajo existen muchas maneras, incluso se puede hacer por software, pero como simplicidad, lo más sencillo es incorporar un cristal u oscilador externo que nos genere 16 MHz.

Se ha escogido el ATtiny84 ya que si usamos un cristal externo para adaptar la frecuencia de uso, con el ATtiny85 nos quedarían solo 4 pines para usar, ya que dos pines son utilizados para incorporar el cristal.

El acondicionamiento de cómo adaptar el ATtiny84 a nuestras necesidades además de una breve introducción a los fuses se encuentra explicado en el anexo 1, “Acondicionando el ATtiny84”,

1.6.2 Protocolo SPI y programadores ISP

Primero distingamos entre ISP y SPI. ISP "*In System Programming*" es un acrónimo que se usa para designar a los programadores, por ejemplo Arduino tiene una opción de trabajar como programador ISP, esto significa que permite la programación de otros microcontroladores, desgraciadamente Arduino para trabajar como ISP, tiene algunos inconvenientes, y a veces tiene problemas para establecer los fuses de los microcontroladores, por lo que se ha visto por experiencia propia, por eso para este proyecto se ha decidido usar inicialmente un programador AVR para establecer lo fuses. SPI "*Serial Port Interface*" es un tipo de comunicación vía serie, es lo que llamamos un protocolo, establece las reglas para la comunicación, desde los niveles de voltajes a las tramas que se han de enviar.

Para realizar una programación SPI, se necesitan una serie de puertos que vienen definidos tanto en el microcontrolador como en el programador, en la imagen 1.4 podremos apreciar que en las patillas determinadas aparecen los términos: MISO, MOSI, AVR y RESET. MISO son las siglas de *Master Input Slave Output,* este pin es la línea de comunicación donde el programador recibe información y en cambio el microcontrolador a programar la recibe, este es el único puerto que trabaja de esta manera, todos los demás envían información desde el programador al microcontrolador. MOSI son las siglas de *Master Output Slave Input*. Este pin envía datos desde el programador al microcontrolador. SCR son las siglas de *Serial Clock,* es la señal de reloj, esta establece la velocidad de transferencia entre el programador y el microcontrolador. RESET, como su propio nombre indica es un pin que establece el inicio de la programación. XTAL1, se refiere a cristal uno y es un pin del microcontrolador conectado al cristal externo. [15]

1.7 Usando Arduino como ISP

Aunque para realizar el acondicionamiento del ATtiny84 hemos usado un programador (ver Anexo 1), a partir de ahora, el programador AVR lo podemos guardar, ya que ahora siempre trabajaremos con Arduino UNO, lo usaremos incluso para subir programas al microcontrolador ATtiny84.

Ahora que ya tenemos el ATtiny84 preparado, veremos cómo subir un programa al microcontrolador. Para ello lo primero que tenemos que hacer es abrir la interfaz de Arduino, en este proyecto se usará la versión 1.0.4 que se puede descargar de: http://arduino.googlecode.com/files/arduino-1.0.4-windows.zip

Lo siguiente que haremos es seleccionar el puerto, y la tarjeta. El puerto normalmente es el COM3 y la tarjeta es "Arduino UNO". A continuación abriremos el ejemplo ArduinoISP, en "Archivo>Ejemplos>ArduinoISP". Y lo cargaremos en la tarjeta.

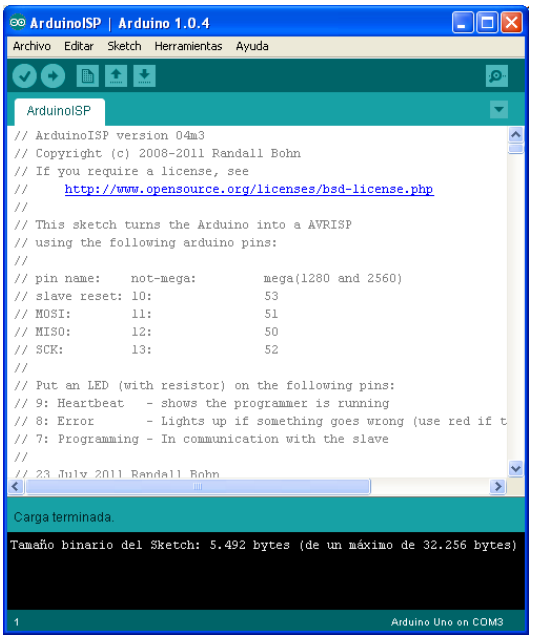


Fig. 1.5 Arduino cargando ejemplo ArduinoISP

Una vez cargado el programa, descargaremos la librería de Arduino-Tiny y la guardaremos en el directorio hardware del directorio donde se ha creado Arduino. Reiniciaremos el programa, y veremos que tenemos más tarjetas para elegir, en concreto la única que nos interesará será “ATtiny84 @ 16 MHz (external cristal; 4.3V BOD”.

Para cargar un programa en la tarjeta debemos conectar Arduino de la manera que se indica en la figura 1.6.

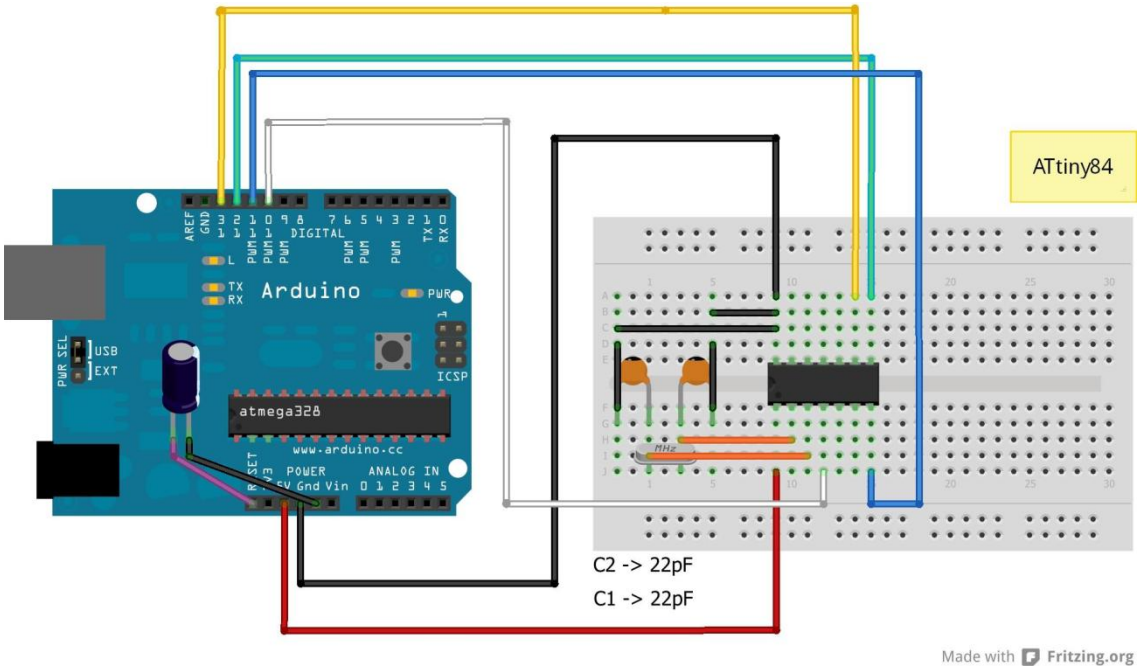


Fig. 1.6 Arduino como ISP conectado a ATtiny84 [16]

Ahora solo tenemos que abrir un programa por ejemplo Blink y elegir el pin de salida, la numeración de los pines en esta librería corresponde a la imagen 1.7.

Fig. 1.7 Nomenclatura de los pines de un ATtiny84 [17]

Como ejemplo usaremos el programa que hemos visto al inicio de este capítulo, pero cambiando el pin 13, por el 2. La configuración de pines se encuentra en la carpeta "\tiny\cores\tiny\core_pins.h".

```
int led = 2;      // Pin 2 -> PB2;
                  // NOTA: La configuración de pines puede variar según
                  // la librería utilizada.

/* Asignamos a una variable "led" el pin que queremos usar como salida
digital. PIN REAL -> PA0 */

void setup() {
  pinMode(led, OUTPUT);      // Asignamos el pin como salida
}
void loop() {
  digitalWrite(led, HIGH);   // Salida del Pin a VCC     -> LED ON
  delay(1000);               // Tiempo de espera 1 segundo
  digitalWrite(led, LOW);    // Salida del Pin a 0V      -> LED OFF
  delay(1000);               // Tiempo de espera 1 segundo
}
```

Fig. 1.8 Programa Blink para ser cargado en un ATtiny84

Seleccionamos la tarjeta "ATtiny84 @ 16 MHz (external cristal; 4.3V BOD" y subimos el programa, Arduino actuará como programador, y no se modificará su contenido, por seguridad, es conveniente poner un condensador electrolítico de 10 uF entre Reset y GND, para evitar que modificar el programa guardado en Arduino. [10]

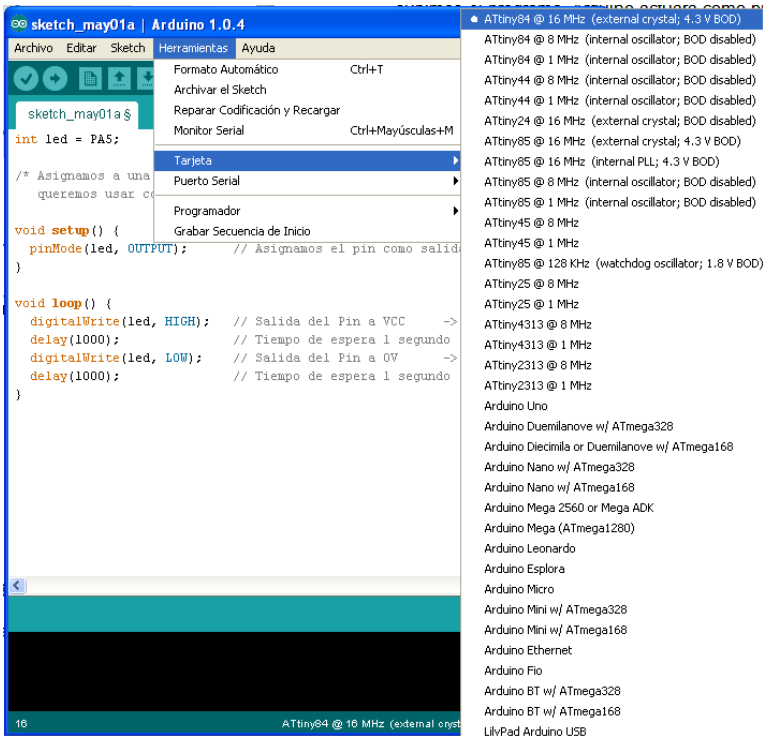


Fig. 1.9 Cargando Blink en un ATtiny84

Una vez cargado el programa y no tenemos ningún error, si conectamos un led entre GND y la salida PB2, veremos que hemos completado con éxito la carga del programa al ATtiny84. Ahora ya somos capaces de trabajar con Arduino en el ATtiny.

1.8 El ADC

Al principio de este capítulo hablábamos sobre la importancia del ADC. En este apartado veremos las características a tener en cuenta en nuestro proyecto y que necesitaremos en capítulos posteriores.

En la tabla 1.3 podemos apreciar las características principales que tiene nuestro conversor analógico digital. De entre todas las especificaciones podemos destacar dos, la resolución y el rango de voltaje.

Tabla 1.3. Especificaciones básicas del ADC del chip ATtiny84 [14]

Parámetro	Valor
Resolución	10 bits
Rango de Voltaje	0 a VCC
Tiempo de conversión	65 a 260 µs
Canales	8
Incertidumbre Máxima	1 LSB

1.8.1 Análisis del ADC del ATtiny84

El conversor analógico digital, es el elemento que transformará la señal analógica que proviene del sensor a una señal digital. Para realizar la conversión analógica a digital, el ADC se basa en comparar la señal de entrada con una serie de tensiones fijas, el resultado de esta comparación es codificado según un valor digital determinado [18]. Para entender esto mejor, diremos que la señal de entrada se cuantifica a una serie de niveles establecidos, y cada uno de estos niveles tiene asignado un número binario. La cantidad de niveles a comparar con la señal de entrada viene determinado por el rango de entrada del ADC, 10 bits tiene una rango de 1024 valores distintos, según la fórmula 1.1.

$$2^n = Rango\ de\ entrada\ ADC;\ \ n, número\ de\ bits\ del\ ADC \quad \textbf{(1.1)}$$

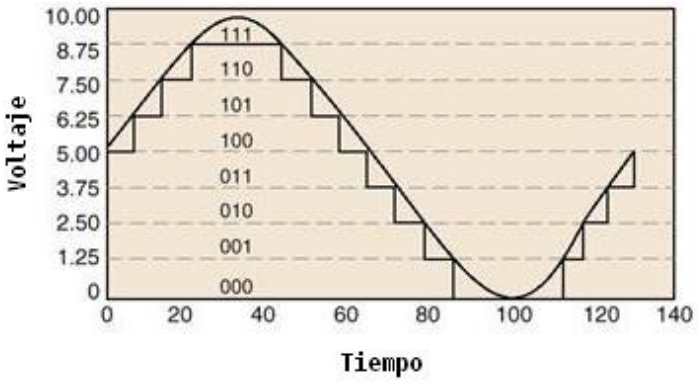


Fig. 1.10 Conversión Analógica a Digital con ADC de 3 bits [19]

Para entender mejor como funciona un ADC, la figura 1.10 ilustra una conversión analógica digital con un ADC de 3 bits, como se puede ver en este caso la señal de entrada que es de 10V se codifica en 2^3 estados diferentes, o lo que es lo mismo 8 estados, desde el 000 al 111.

Si establecemos un valor de VCC de 5V, podemos saber que intervalo de tensiones de entrada corresponde a cada estado, haciendo la operación 1.2, tenemos que cada 4.88 mV el ADC detectará un cambio de estado, este dato es la resolución del ADC. [20]

$$Resolución\ del\ ADC = {}^{VCC}/_{2^n} = {}^{5V}/_{2^{10}} = 4.88\ mV\ ;\ \ n = bits\ del\ ADC \quad \textbf{(1.2)}$$

1.9 Diseño de la unidad remota

1.9.1.1 La Batería

En este punto tenemos un desafío importante, si nos hemos dado cuenta hemos hablado de trabajar con un ATtiny84 de forma remota, pero para ello necesitaremos trabajar con una fuente de alimentación portátil, una batería o una pila.

Bien, para alimentar el sistema de la unidad remota, se va utilizar una batería de 3.7V de Litio Polímero de 200mAh. ¿Por qué? Básicamente porque la mayoría de helicópteros radio control más baratos del mercado incorporan una batería como esta.

Para este proyecto se utilizará un helicóptero de radio control de desguace, como base de pruebas, se utilizarán principalmente el bloque motor y las baterías, como comprendo que no todo el mundo tiene en su casa un helicóptero radio control para ser desguazado, una de las opciones más económica se expone en la tabla 1.4.

Tabla 1.4. Precio del producto radiocontrol utilizado para piezas

Distribuidor	Referencia	Precio
www.amazon.es	Fun2get-Falcon	17,12 € *

* Precio sin IVA, día: 1/05/2013

Comprar este modelo de radiocontrol es más económico incluso que comprar las piezas por separado, aunque estos helicópteros no pueden soportar cargas de más de unos cuantos cientos de gramos, nos servirá como base de operaciones, y como conejillo de indias para nuestro proyecto.

1.9.1.2 Conversor de carga

Los conversores de carga son elementos capaces de transformar una tensión y una intensidad en entrada en una tensión y una intensidad de salida diferente, normalmente siendo la tensión de salida mayor que la de entrada, en caso contrario, estaríamos hablando de reguladores de tensión.

En nuestro proyecto, tenemos una batería Li-Po capaz de dar 3.7V y 750 mA a y con una capacidad de 200mAh. Estos valores debemos transformarlos en una tensión de salida regulada a 5V y a una intensidad de salida suficiente para alimentar a todos los elementos del circuito, microcontroladores, sensores, motores, etc.

En este punto debemos elegir nuestro conversor de carga o también llamado conversor de potencia, ya que la potencia debería ser teóricamente igual entre la entrada y la salida, aunque en una conversión siempre existe una perdida.

Para elegir nuestro conversor de carga, debemos tener en cuenta que estamos hablando de un circuito aumentador, de 3.7V a 5V y como ya se ha

especificado en la introducción del trabajo con un encapsulado que nos permite trabajar en protoboards.

Para elegir el producto más adecuado a nuestras necesidades, se ha realizado una comparativa de productos, la cual se expone en la tabla 1.5.

Tabla 1.5. Comparativa de conversores de carga

Producto / REF	Tensión de entrada	Tensión de salida	Intensidad de salida	Precio
LT1073CN8	1,5V	5V	40 mA	5,72 €
LT1303CN8	2 V	5V	200 mA	3,29 €
MAX664CPA+	-9V	5V	40 mA	4,76 €
MAX756CPA+	2,5V	5V	200 mA	4,98 €
Filtro previo de la búsqueda: Conversor de carga con una tensión de entrada no superior a 4V y una salida a 6V. Encapsulados escogidos: DIP.				

Como podemos observar, tenemos dos opciones claramente, el LT1303CN8 y el MAX756CPA. Para elegir entre uno y otro lo mejor es ver los elementos adicionales que integran cada uno según sus datasheets.

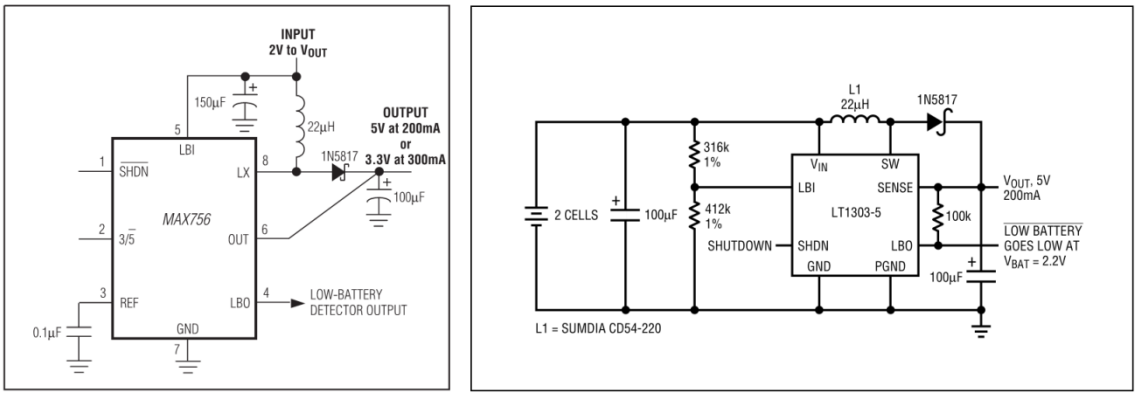


Fig. 1.11 Comparativa MAX756 vs LT1303 [21]

Como se pueda observar en la figura 1.11 existen elementos comunes, como la inductancia o el diodo schottky, pero el que menor número de elementos presenta es el MAX756, aunque para ser justo es LT1303 tiene una salida variable lo que para otros proyectos puede ser muy útil.

El circuito del conversor de carga se ha modelo usando Eagle 6.4.0 y puede verse el circuito completo, en la figura 1.12. A parte se ha creado una tabla (1.6) con los elementos adicionales necesarios para realizar el circuito, estos elementos se pueden encontrar en cualquier tienda de electrónica, en nuestro caso hemos añadido las referencias de http://es.farnell.com.

Tabla 1.6. Tabla de los componentes del conversor de carga MAX756

Producto / REF/Link	Descripción	N	Precio Unitario
MAX756CPA+	Conversor de carga DC/DC	1	4,980 €
1N5817	Diodo Schottky	1	0,153 €
11R223C	Inductor de 22 uH	1	0,370 €
MCDS02	Interruptor DIL	1	0,770 €
MCGPR10V107M5X11	Cond. electrolítico 100 uF, 10 V	1	0,074 €
EEUFR1A151	Cond. electrolítico 150 uF, 10 V	1	0,099 €
MCRR25104X7RK0050	Cond. cerámico 100 nF 50 V	1	0,330 €
** SPC15494	Zócalo 8 pines DIP	1	0,124 €
** SPC15496	Zócalo 14 pines DIP	1	0,155 €
** 1776275-2	Bloque terminal 2 vías	1	0,250 €
** M22-2510205..	Conector Vertical, Pin Macho	1	0,192 €
Precio sin IVA, con los elementos optativos:			7,497 €

** Elementos optativos, se pueden adquirir, para facilitar la extracción del chip, o de la batería, o en el caso del conector vertical, para tener un acceso donde hacer mediciones más fácilmente, precio total del módulo sin elementos optativos: 6,776 €, sin IVA.

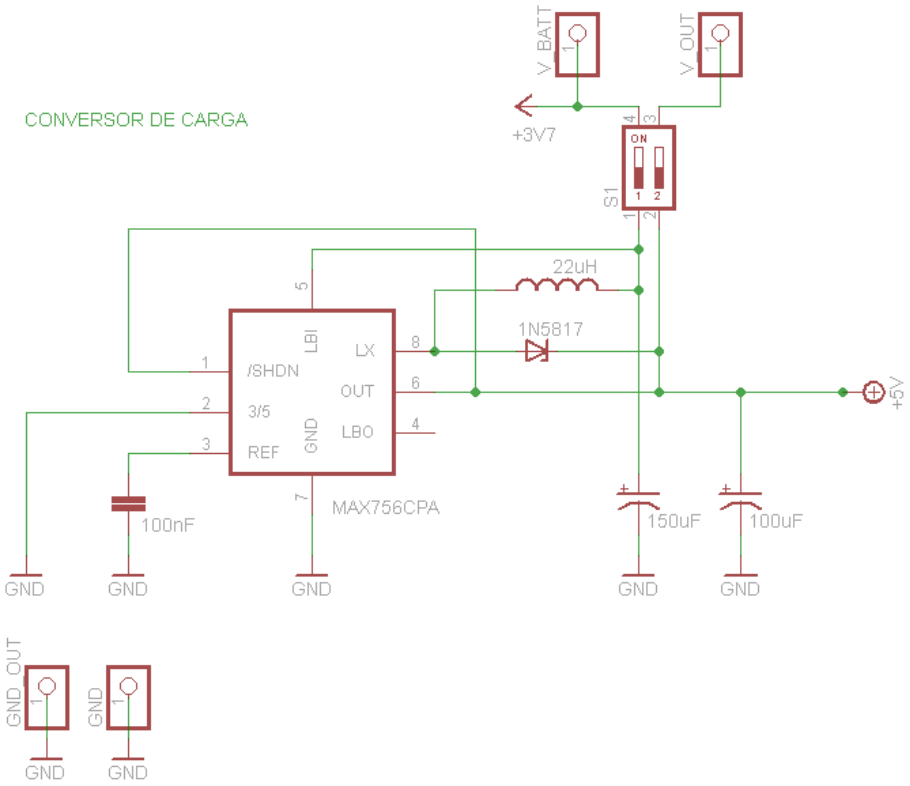


Fig. 1.12 Conversor de carga MAX756CPA

NOTA AL LECTOR: En el anexo 3, “Resultados experimentales” se pueden ver datos experimentales y material adicional sobre el acondicionamiento de la unidad de procesamiento.

1.10 Esquema del circuito de la unidad de procesamiento

La unidad de proceso, está completada. Por un lado tenemos Arduino UNO y por otro tenemos un microcontrolador ATtiny84 conectado a una batería a través de un conversor de carga que nos proporciona 5V.

Tabla 1.7. Tabla de los componentes de la unidad de proceso

Producto / REF/Link	Descripción	Cant.	Precio Unitario
Arduino UNO REV 3	Plataforma Arduino	1	0 €*
ATTINY84-20PU	Microcontrolador ATtiny84	1	3,010 €
D220G20C0GH63J5R	Condensador 22pF	2	0,181 €
XT49S16M	Cristal 16 MHz	1	0,55€ €
Conversor de carga	Circuito completo**	1	7,497 €
Coste completo de la unidad de procesamiento			11.419 €

* Material cedido por la universidad.
** Circuito completo con elementos optativos incluidos.

El esquema del módulo de la unidad de procesamiento se presenta en la figura 1.13, Arduino UNO no se ha representado, al ser una placa propia totalmente integrada y funcional.

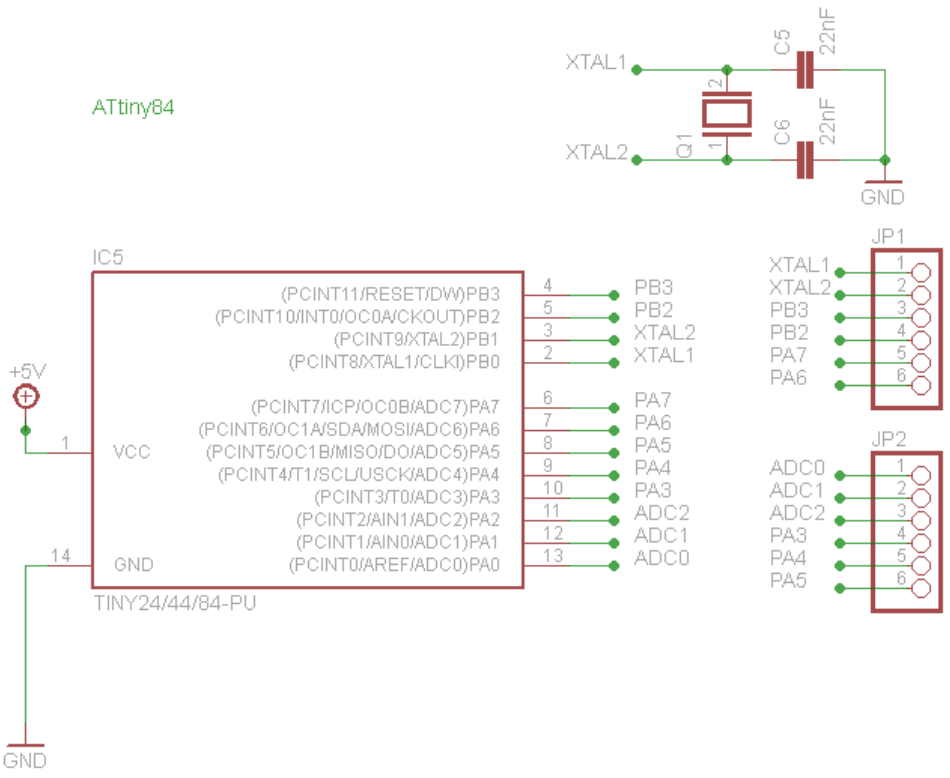


Fig. 1.13 Acondicionamiento del ATtiny84 para uso remoto, ver NOTA.

NOTA: En delante en la memoria, para facilitar la representación de cada módulo la señal +5V, corresponderá a VCC que será la salida que nos proporciona el conversor de carga MAX756.

CAPÍTULO 2. ACONDICIONAMIENTO DE SENSORES Y ACTUADORES

Una vez visto el capítulo de la unidad de procesamiento, ya podemos empezar a seleccionar y adaptar nuestros sensores y actuadores al sistema. En el siguiente capítulo se van a condicionar tres tipos diferentes de sensores: temperatura, efecto hall e infrarrojos; además de un actuador, el motor de continua.

Normalmente, la elección de los sensores en el ámbito aeronáutico, siempre suele recaer sobre acelerómetros y giroscopios, ya que estos nos permiten conocer la actitud de vuelo. En este proyecto no se ha destacado por este tipo de sensores por dos motivos, primero, estos sensores suelen encontrarse en encapsulados no admisibles con los requisitos previos establecidos y segundo, la gran mayoría de estos sensores suelen dar una señal digital en su salida, por lo que el acondicionamiento previo suele ser inexistente.

2.1 Teoría de acondicionamiento

2.1.1 Definición de sensor

Primero definiremos que es un sensor. Un sensor, también llamado transductor, es un dispositivo capaz de transformar una magnitud física a una magnitud eléctrica, normalmente una señal analógica, aunque no necesariamente, ya que existen sensores que ofrecen una salida digital, aunque no son estudiados en este proyecto. Cuando hablamos de señal analógica, nos referiremos a una señal con dos componentes: amplitud y tiempo, básicamente tendremos una determinada amplitud en un tiempo definido.

2.1.2 Parámetros básicos de un sistema de medida

Antes de empezar con la elección de los sensores, debemos aprender una serie de conceptos que son esenciales en cualquier diseño, estos son: el margen, la sensibilidad, la resolución, la exactitud, la repetibilidad y el margen dinámico.

El margen o *range* es el campo o intervalo de medida que queremos analizar, por ejemplo, si hablamos de un sensor de temperatura, el margen podría ser de 0 a 100 ºC. La sensibilidad es la relación entre la magnitud de salida (eléctrica) y la de entrada (temperatura, campo magnético, etc.), por ejemplo, en el caso de temperatura se suele expresar como V/ºC o mV/ºC, cuando la sensibilidad es constante hablaremos de sensores lineales, y la sensibilidad será la pendiente en la curva de calibración. La resolución o *resolution* es la variación más pequeña que podemos o queremos detectar, 0.1ºC, 1ºC, etc. La exactitud o precisión, en inglés *accurary*, es la diferencia entre el valor medido y el valor de referencia usado para calibrar, la exactitud siempre será peor que la resolución, en el caso ideal será igual. La repetibilidad o en inglés *precision* (no confundir con precisión) es la diferencia entre los valores que va adquiriendo

sucesivamente la salida del sensor bajo unas mismas condiciones de medida, si nuestra repetibilidad es baja suele ser debido a errores aleatorios en la circuitería del sensor (ruido, factores temporales, etc.) y pueden ser eliminados (o reducidos) haciendo la media entre n medidas sucesivas o filtrando. [20] [22].

A parte de los parámetros que hemos visto en los sensores otro parámetro fundamental es el margen dinámico. El margen dinámico, MD, o en inglés, *dinamic range*, es la relación entre el margen y la resolución, tal y como se establece en la fórmula 2.1.

$$MD = {Margen\ de\ medida}/{Resolución} \qquad \text{[20]}\textbf{(2.1)}$$

El margen dinámico nos proporciona la cantidad de intervalos mínimos que necesitamos para hacer el acondicionamiento. Este número siempre tiene que ser menor que 2^n, siendo n la resolución de bits de nuestro ADC. [23] Como vemos este parámetro será muy importante para hacer los cálculos de acondicionamiento. El margen dinámico también se suele expresar en decibelios, para ello, el margen dinámico debe ser calculado tal y como indica la fórmula 2.2.

$$MD = 20\log\left({Margen\ de\ medida}/{Resolución}\right) \qquad \text{[20]}\textbf{(2.2)}$$

Cuando hablemos de sensores, además de los parámetros mencionados, existen muchos más, entre ellos se puede resaltar: el ruido, la temperatura de funcionamiento, la tensión de voltaje permitida, etc. Todos estos parámetros se pueden encontrar en el datasheet del fabricante y pueden llegar a ser críticos en el diseño de un sistema.

2.1.3 Elementos básicos en el acondicionamiento

Debido a la limitación de páginas establecida para la presente memoria, el tema de circuitos de acondicionamiento, se abordará en el Anexo 2, en el capítulo “Elementos básicos”. En este apartado se establece la teoría básica de dos elementos muy importantes en el tema del acondicionamiento de sensores: el amplificador operacional y el transistor BJT.

2.2 Sensor de temperatura

2.2.1 Teoría

El primero de los sensores que vamos a ver en este proyecto es el sensor de temperatura. Existen diferentes tipos de sensores de temperatura, principalmente tenemos dos grandes tipos: los resistivos, y los sensores generadores; este tipo de sensores se escogen mayoritariamente en función de parámetros como el rango de temperatura a medir, la resolución y la linealidad.

2.2.1.1 Sensores de temperatura resistivos

Los sensores de temperatura resistivos, tal como su nombre indica, se basan en la variación de su resistencia en función de la temperatura a la que están sometidos. Tenemos dos grandes tipos dentro de este conjunto, los RTD y los Termistores. Los termistores a su vez se pueden clasificar en NTC, si su coeficiente de temperatura es negativo y PTC si su coeficiente de temperatura es positivo. Una representación de la simbología utilizada en los sensores resistivos se puede ver en la figura 2.1.

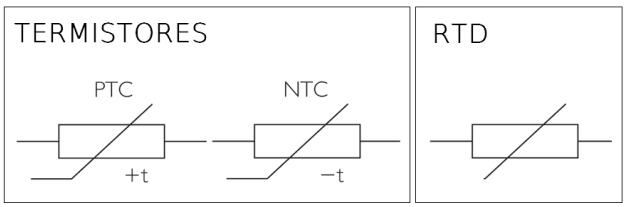


Fig. 2.1 Simbología de los sensores de temperatura resistivos [24]

Para no profundizar mucho en el tema, diremos que la diferencia básica entre los RTD y los Termistores, es que los primeros se basan en la variación de la resistencia de un conductor y los segundos se basan en semiconductores, que según la temperatura varia el número de portadores. Existe otra diferencia importante entra los RTD y los termistores, su salida. La salida de los RTD es lineal y su función característica se presenta en la fórmula 2.3, donde R_0 es la resistencia a la temperatura de referencia, T es el incremento de temperatura respecto a la temperatura de referencia y α_1 es un coeficiente que depende del material, por ejemplo en el platino este valor suele ser de 0,0038 (Ω/Ω)/ºC.

$$R = R_0(1 + \alpha_1 \mathrm{T}) \quad \textbf{(2.3)}$$

En cambio los termistores presentan una salida exponencial, tal como indica la fórmula 2.4, donde B es la temperatura característica del material y T_0 es la temperatura en Kelvins.

$$R = R_0 e^{[\pm B(1/T - 1/T_0)]} \quad \textbf{(2.4)}$$

Cabe destacar que los sensores de temperatura resistivos presentan una muy buena exactitud, siendo mayor en los termistores que en los RTD, además de esto los termistores presentan unas características muy interesantes, que veremos a continuación y que pueden ser muy útiles en futuros proyectos.

Dentro de los termistores, los NTC pueden ser usados en circuitos como limitadores de corriente, esto representa que el NTC puede actuar como elemento de protección poniendo el NTC en serie con el circuito. Esto se basa en que la resistencia al principio en este elemento es alta de manera que la corriente se ve limitada, protegiendo al circuito en serie, una vez el NTC se ha estabilizado permite el paso de una corriente de una manera estable.

Los PTC en cambio pueden ser usados como elementos de protección en sistemas de motores ya que estos al principio presentan una baja resistencia y permiten pasar una corriente elevada en el arranque de los motores y después al paso del tiempo, cuando se calienten debido a la intensidad de circulación la resistencia del elemento aumenta de forma exponencial y deja pasa un valor muy pequeño de intensidad. [20][23][25]

2.2.1.2 Sensores generadores

Este tipo de sensores generadores a diferencia de los resistivos se basan en la generación de una señal eléctrica sin necesidad de una fuente de alimentación.

En concreto estos sensores de temperatura se basan en los principios físicos del efecto Peltier y el efecto Thompson. El efecto Peltier se basa en que la unión de dos metales diferentes que debido al cambio de temperatura entre ellos se genera una circulación de corriente proporcional al cambio de temperatura. El efecto Thompson se basa en que la circulación de corriente en un conductor produce una absorción o liberación de calor por parte del conductor.

Para explicar un poco el párrafo anterior, digamos que el efecto Peltier es reversible, si existe cambio de temperatura entre la unión de dos metales existen una circulación de corriente, y al revés si circula una corriente entre la unión de dos metales existirá un cambio en la temperatura.

Entre los sensores generadores los más famosos o los más utilizados en el mundo de la aviación son los termopares. Los termopares tienen la ventaja de poder medir un margen de temperaturas muy elevados, superiores incluso a los 1000 ºC, aunque su respuesta en no lineal, su uso para medir temperaturas en motores es muy común.

Fig. 2.2 Simbología de un termopar

2.2.1.3 Sensores integrados

Hoy en día existen sensores de temperatura encapsulados en diferentes formatos y que nos dan una salida lineal en tensión en función de la temperatura, uno de los más "famosos" es el LM35, un circuito integrado de la empresa Texas Intruments® que nos proporciona una salida de 10mV por grado Centígrado.

Estos sensores integrados se encuentran en diferentes tipos de encapsulados y suelen basarse en sensores resistivos, tanto RTD como en termistores. La gran ventaja que presentan estos dispositivos es su simplicidad en el diseño, ya que no necesitan componentes adicionales más allá del amplificador operacional.

2.2.2 Elección del sensor

En este caso la elección del sensor se basará en los especificados en la tabla 2.1 además de los criterios ya especificados al inicio de este proyecto.

Tabla 2.1 Parámetros de elección del sensor de temperatura

Parámetro	Valor
Margen	5 a 45 ºC
Rango de Funcionamiento	0 a 5 V
Resolución	Mejor Posible
Precio	Menor Posible
Simplicidad del diseño	Menor Posible

Como podemos observar, en este primer sensor los factores de diseño son bastantes simples, lo que es claramente intencionado, para intentar una adaptación al ADC lo más precisa posible, esto como se verá más adelante no será posible en el resto de sensores a acondicionar en este proyecto.

Para la elección del sensor, nos basaremos en una comparativa entre los diferentes sensores que podemos encontrar, esta comparativa se puede ver en la tabla 2.2.

Cómo elección se ha escogido el "LM35DZ", para realizar este proyecto. La razón de escoger este sensor a favor de otro, es por su linealidad y su bajo coste. Cabe destacar que como una segunda opción o alternativa, se hubiese escogido un sensor resistivo NTC, por su bajo coste, y aunque este tipo de sensor no se ha escogido por no ser lineal, se puede linealizar basándose en diversas configuraciones de pseudo-puentes o puente de *wheatstone*, por lo que para otros proyectos, donde el precio sea más crítico se puede anteponer a sensores integrados.

Tabla 2.2 Parámetros de elección del sensor de temperatura

Sensores Integrados				
Producto / REF	Tipo Sensor	Rango de Temperaturas	Sensibilidad	Precio
LM35DZ	Integrado - Lineal	2 – 100 ºC	10mV/°C	1,89 €
AD22100KTZ	Integrado - Lineal	-50 – 150 ºC	22,5mV/°C	4,25 €
LM35CAZ	Integrado - Lineal	-40 – 110 ºC	10mV/°C	4,41 €
Filtro previo de la búsqueda: Sensor de temperatura integrado, Máximo ± 0.5 ºC de precisión.				
Sensores RTD				
Producto / REF	Tipo Sensor	Rango de Temperaturas	Resistencia	Precio
DM-513	RTD PT100	-50°C - +500°C	100 Ω	5,69 €
DM-504	RTD PT1000	-50°C - +500°C	1 kΩ	6,83 €
Filtro previo de la búsqueda: PT100, PT100. Menor Precio.				
Termistores				
Producto / REF	Tipo Sensor	Rango de Temperaturas	Resistencia	Precio
NTCLE305E4502SB	NTC	-40 ºC – 125 ºC	5kΩ	0,75 €
NTCLE203E3103GB0	NTC	-40 ºC – 125 ºC	10kΩ	0,81 €
Filtro Previo: Termistores, Tolerancia: ± 0.5 ºC.				

2.2.3 Acondicionamiento del sensor

El acondicionamiento de este sensor es bastante simple, como su salida es proporcional a la temperatura, lo único que tendremos que hacer es adaptar nuestra salida al conversor ADC de nuestro ATtiny84.

Para adaptar la salida se ha escogido un amplificador operacional RAIL-to-RAIL, que puede ser operado entre 0 y 5V. El operacional elegido es el TLC2272. Se ha escogido este tipo de operacional por intentar buscar la máxima resolución con el mínimo número de elementos.

Tabla 2.3 Características básicas de un TLC2272

Producto / REF	Tipo Amplificador	Operacionales integrados	Precio
TLC2272AIP	Rail-to-Rail, Low Noise.	2	2,48 €

Para calcular la resolución teórica que tendrá nuestro sistema podemos utilizar la fórmula del margen dinámico:

$$MD = {Margen\ de\ medida}/{Resolución} = 45\ ºC - 5ºC\ /_{X\ ºC} = 1024 \qquad \textbf{(2.5)}$$

En realidad su utilizásemos todo el margen que nos proporciona el ADC, seriamos capaces de adquirir resoluciones teóricas de hasta 0,05 ºC. Al adaptar el margen de medidas a todo el ADC, en verdad lo que estaremos consiguiendo es aumentar la resolución del sistema.

Lo más fácil para conseguir adaptar la salida del sensor al ADC, es aumentar la salida del sensor por un factor x, para adaptarlo a todo el rango del ADC y utilizar así las 1024 posiciones, o en su defecto el mayor número de posiciones posibles.

La sensibilidad del sensor LM35DZ es de 10 mV/ºC, por lo que en el menor valor posible de salida (5 ºC) tendremos una salida de 50mV y en el caso del máximo valor (45 ºC) tendremos una salida de 450 mV, que adaptaremos a 5 V mediante una ganancia de 11. Cómo se ha podido ver en el anexo 2, "Elementos básicos", la salida de un operacional no inversor es de la siguiente forma:

$$V_{out} = V_{in}\mathrm{G} = V_{in}(^{R2}/_{R1} + 1) \qquad \textbf{(2.6)}$$

Para conseguir una ganancia 11, lo único que tenemos que hacer es conseguir una relación R2/R1 de 10. No se ha considerado ajustar la salida menor (5º), hasta 0 V ya que sería necesario incrementar el número de elementos del sistema sin conseguir un aumento realmente significativo de resolución.

La adaptación de nuestro sensor de temperatura, se representa en la figura 2.3. La salida de este sensor se ha asignado al pin "PA0" del ATtiny84, en el esquemático llamado "ADC0".

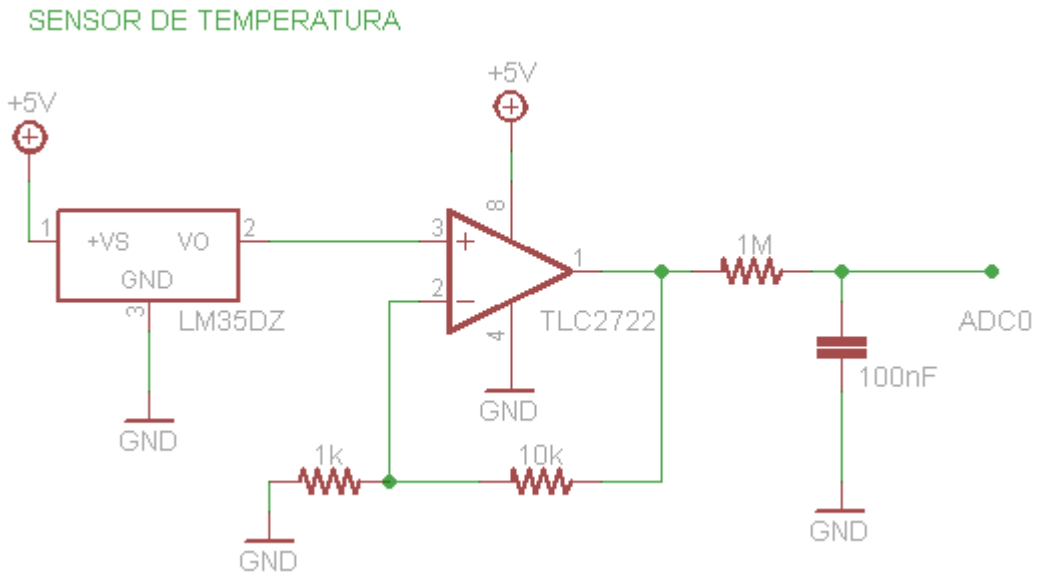


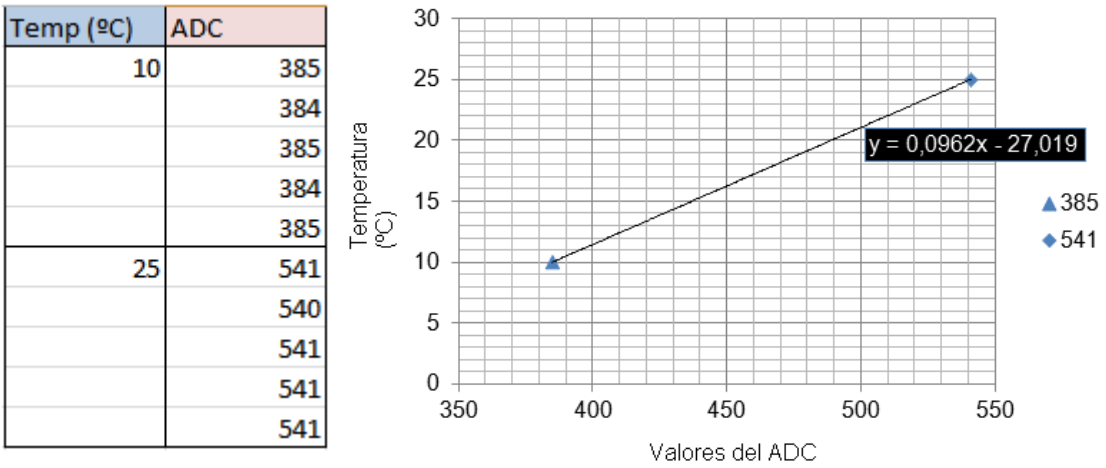
Fig. 2.3 Acondicionamiento del sensor LM35DZ

2.2.4 Calibración

La calibración del sensor es muy sencilla, debida a que nuestro sensor es lineal, lo único que tenemos que hacer es obtener un mínimo de 2 puntos con una referencia más precisa, y determinar los coeficientes de la recta de calibración.

Para establecer una curva de calibración lo primero que debemos hacer es comparar nuestra salida del ADC con algún instrumento con una mejor resolución. Solo falta poner el sensor a dos temperaturas diferentes y podemos calibrar el sistema, usando estos dos puntos de medida con otros dos puntos de medida calibrados.

Para obtener la curva de calibración existen muchos métodos y sistemas, el más utilizado o al menos el más conocido, es el método de mínimos cuadrados. El método de mínimos cuadrados, es una aproximación matemática a una curva conocida. Para utilizar este método simplemente tendremos que usar un simple programa de cálculo para calcular el parámetro de la curva de calibración. El camino más sencillo es usar una hoja de Excel.



Temp (ºC)	ADC
10	385
	384
	385
	384
	385
25	541
	540
	541
	541
	541

Fig. 2.4 Curva de calibración y datos de calibración del sensor LM35DZ

2.2.5 Resumen

Tabla 2.4. Tabla de los componentes del sensor de temperatura

Producto / REF/Link	Descripción	Cant.	Precio Unitario
LM35DZ	Sensor de Temperatura	1	1,89 €
TLC2272AIP	Amplificador Operacional	1	2,48 €
MCRR25104X7RK0050	Cond. cerámico 100 nF 50 V	1	0,330 €
MCF 0.25W 1M	Resistencia 1M 1/4W	1	0,018 €
MCF 0.25W 1K	Resistencia ¼ W 1K	1	0,018 €
MCF 0.25W 10K	Resistencia ¼ W 10K	1	0,018 €
Coste completo del sensor de temperatura			4,754 €

2.3 Sensor de efecto Hall

Existen diferentes tipos de sensores magnéticos, entre ellos las magneto resistencias y los sensores de efecto Hall son los más utilizados en este campo de medición. Para realizar este proyecto se han elegido los primeros, los sensores de efecto Hall, debido a su bajo coste.

Con la adaptación de este sensor se pretende crear una brújula magnética capaz de detectar el norte magnético terrestre. El campo magnético varía en función de la localización geográfica, en concreto en Barcelona es de 45 398,5 nT, o lo que es lo mismo 0,45398 G. Para calcular el campo magnético terrestre en función de la localización se recomienda visitar: http://www.ngdc.noaa.gov/geomag-web/#igrfwmm

2.3.1 Teoría

Los sensores de efecto Hall, son sensores que son capaces de medir campos magnéticos en su área de medición. Este efecto se basa en que una corriente que circula en un conductor y que está inmersa dentro de un campo magnético, perpendicular a la misma, crea un desplazamiento de los portadores de carga, estos portadores se agrupan en los lados del conductor o semiconductor y aparece una diferencia de potencial que puede ser medida y es proporcional al campo magnético.

2.3.2 Elección del sensor

En la elección del sensor de este sensor, se han elegido dos compañías líderes en la fabricación de este tipo de sensores, Allegro Microsystems® y Honeywell®. La comparativa de los diferentes tipos de sensores, se representa en el anexo 4, “Comparativa sensores efecto Hall”, debido al tamaño de la tabla comparativa.

La elección en este sensor ha sido el A1301, que presenta una sensibilidad menor que el SS495A1 pero tiene un coste inferior. La sensibilidad de este sensor es de 2.500 mV/G. Cómo nota al lector, cabe destacar que el sensor más apropiado para los fines de este proyecto sería el SS494B, pero lamentablemente este sensor no estaba disponible en la web de Farnell®, lo que no se escogió para no contradecir uno de los requisitos del proyecto, el de la fácil disponibilidad.

2.3.3 Acondicionamiento

Según el campo a calcular (0,45 G) y la sensibilidad del sensor (2,500 mV/G), tenemos que ser capaces de detectar cambios de 1 mV a la salida de nuestro sensor.

La gran mayoría de este tipo de sensores, presentan a la salida VCC/2 más la medida del campo, como vemos el propio sensor nos aporta un offset, que debemos eliminar para realizar las mediciones.

Para acondicionar este tipo de sensores, no nos basta con un operacional normal, como hemos usado para realizar el acondicionamiento del sensor de temperatura, tenemos que ir un paso más allá. La mejor elección para adaptar un sensor cuyo campo a medir es muy pequeño, son los amplificadores operacionales de instrumentación.

Los amplificadores de instrumentación son amplificadores que presentan en un su entrada una alta impedancia, además de una ganancia estable que se consigue con una única resistencia. También un factor muy importante es que este tipo de amplificadores, presentan unos niveles de ruido muy bajos. Los amplificadores de instrumentación pueden ser construidos con la suma de otros operacionales más simples, pero debido a que queremos ahorrar el máximo de espacio, nos hemos decantado por el amplificador de instrumentación INA122PA.

Además de proporcionarnos todas las ventajas mencionadas en el párrafo anterior, este tipo de amplificadores nos presentan un salida de la siguiente forma:

$$V_{out} = (V_+ - V_-)G \qquad \textbf{(2.7)}$$

Lo que nos facilita enormemente el trabajo, ya que V_+ puede ser la salida del sensor, y V_- un divisor de voltaje que presente el mismo offset del sensor, por lo que a la salida solo tendremos amplificada la señal del campo magnético detectado.

Debido a que si realizamos un divisor de tensión con dos resistencias, podemos inducir a un error importante en la salida si este divisor no presenta el 50% exacto de nuestro VCC, se ha decidido poner un potenciómetro en la entrada V_-, para calibrar manualmente la salida del divisor y así ajustar el máximo posible la salida del sensor.

En cuanto a la ganancia del operacional, deberemos trabajar en el orden de ganancia 1000. Un pequeño resumen del estudio realizado para la adaptación de este sensor se encuentra en el anexo 3, “Resultados experimentales”. Para realizar las pruebas se ha conectado el circuito representado en la figura 2.5 conectado a la salida “A0” del Arduino UNO y un pequeño programa en LabVIEW que representaba el histograma de las medidas que iba adquiriendo Arduino. Dichos programas se pueden encontrar en el anexo 3, “Resultados experimentales”.

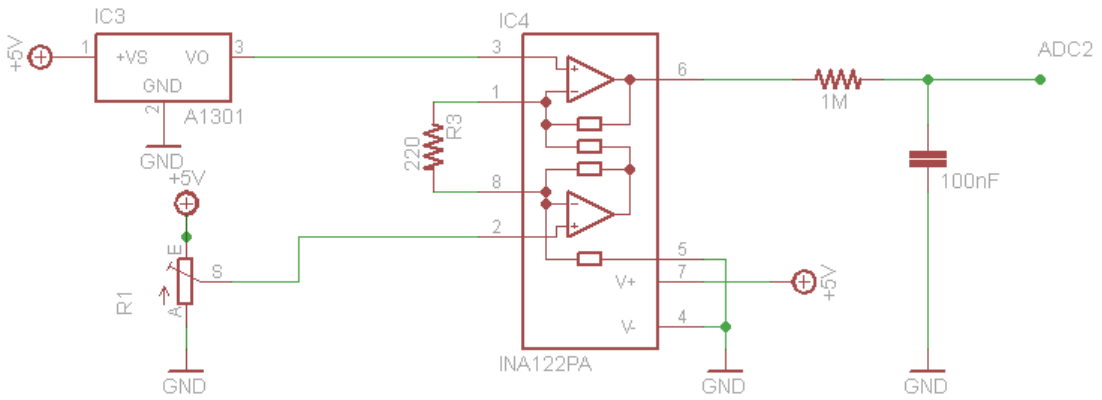


Fig. 2.5 Acondicionamiento del sensor A1301

2.3.4 Calibración

La calibración de este sensor se basa en adquirir datos en los 360º y el valor que tenga una intensidad máxima (el mayor valor en el ADC) corresponderá con la dirección del Norte magnético terrestre. Para encontrar el norte magnético el sistema compara el valor de máxima intensidad con el valor actual del ADC.

2.3.5 Resumen

Tabla 2.5. Tabla de los componentes del sensor de efecto Hall

Producto / REF/Link	Descripción	Cant.	Precio Unitario
A1301EUA-T	Sensor de Efecto Hall	1	1,61 €
INA122PA	Amplificador Instrumentación	1	6,30 €
MCCFR0W8J0221A20	Resistencia 1/4 W 220	1	0,006 €
3306P-1-103 *	Potenciómetro 10K	1	0,38 €
MCRR25104X7RK0050	Cond. cerámico 100 nF 50 V	1	0,330 €
MCF 0.25W 1M	Resistencia 1M 1/4W	1	0,018 €
Coste completo del acondicionamiento del sensor de efecto Hall			8,644 €

2.4 Infrarrojos

2.4.1 Teoría

Primero de todo, definamos que es la luz infrarroja. La luz infrarroja es un tipo de radiación electromagnética que tiene una longitud de onda entre 1mm y los 700nm. Esta longitud de onda no permite que este tipo de radiación sea visible para el ojo humano, ya que este solo es capaz de percibir la radiación que tenga una longitud de onda entre 400 a 700 nm por lo que la radiación infrarroja es la que se encuentra justo debajo (si hablamos de frecuencia) de la luz visible en el espectro electromagnético. [26][27],

Existen diferentes tipos de sensores infrarrojos, pero primero tenemos que dividir entre emisores de luz infrarroja y receptores de luz infrarroja. En el primer caso, los emisores de luz infrarroja, son dispositivos, muy parecidos a los led's pero que son capaces de transmitir en el espectro descrito anteriormente. En el segundo caso existen varios sensores, sensibles a la luz infrarroja, básicamente existen dos grandes grupos: los LDR y los fotodiodos.

Los LDR, *Light Dependent Resistor*, son sensores resistivos que varían su resistencia en función de la intensidad de luz recibida, en este proyecto no se analizarán estos sensores porque su variación resistiva es muy lenta comparada con los fotodiodos, por lo que no son muy recomendados si queremos utilizarlos para transmitir información. Por otro lado, los fotodiodos son semiconductores basados en una unión PN y son sensibles a la luz visible y a la infrarroja, su funcionamiento se asemeja también al de un diodo, pero con la diferencia que la intensidad de circulación interna depende de la luz recibida. Estos sensores se basan en la variación de la corriente que circula a través de ellos, la variación de corriente es proporcional a la cantidad de luz recibida. Cabe destacar que los fotodiodos funcionan con polarización inversa, este dato es importante, porque si colocamos mal un fotodiodo, simplemente no funcionará. [28]

2.4.2 Elección del emisor

Para el primer caso, analizaremos la elección y el diseño del emisor de infrarrojos. Esta tarea es relativamente sencilla, porque el factor más característico de estos dispositivos, es la intensidad máxima que son capaces de soportar, por lo que si Arduino nos limita a 50 mA, este será nuestro valor de intensidad operativa. Cómo siempre usaremos una tabla para hacer la comparativa entre las diferentes elecciones de diseño. En la elección del diseño podríamos haber elegido un emisor por debajo de los 50 mA, ya que un pin digital de Arduino nos proporciona cómo máximo 40 mA, por lo que se podría haber conectado directamente, en nuestro caso hemos considerado mejor optimizar al máximo en este punto, usando la máxima corriente que nos proporciona Arduino UNO, que es de 50 mA, en el pin de 3.3V. Para minimizar la placa, sería posible conectar directamente el emisor a una salida digital, aunque perderíamos potencia de emisión. En la tabla 2.6 tenemos la comparativa de los diferentes sensores infrarrojos.

Tabla 2.6 Comparativa de emisores de infrarrojos

Producto / REF	Ancho de Banda	Intensidad de circulación*	Angulo de transmisión	Intensidad radiante	Precio
OFL-3102	940 nm	20 mA	30º	14mW/Sr	0,070 €
OFL-5102	940 nm	20 mA	10º	15mW/Sr	0,070 €
L-934F3C	940 nm	20 mA	50º	3mW/Sr	0,015 €
CQY36N	950 nm	50 mA	55º	1.5mW/Sr	0,460 €
OPE5685	850 nm	50 mA	22º	50mW/Sr	0,680 €
OPE5587	880 nm	50 mA	10º	50mW/Sr	0,700 €
Filtro previo de la búsqueda: Emisores Infrarrojos, Intensidad de circulación máxima, “forward current”: 50 mA. * Esta es la circulación de circulación operativa, no la máxima admisible.					

Nuestra elección, tanto para conectar directamente a un pin digital (40mA), como para conectar al pin 3.3V (50 mA) usaremos el emisor OPE5685, por ser el que tiene más intensidad radiante, otra alternativa aceptable sería usar el CQY36N si necesitamos un ángulo de transmisión mayor.

2.4.3 Elección del receptor

En la elección del receptor, se ha optado por elegir entre los diferentes tipos de receptores infrarrojos, eligiendo finalmente la opción del fotodiodo, pero no del componente propio, ya que a través de prueba y ensayo se ha comprobado que la luz artificial puede aceptar en su comportamiento, sino que finalmente se ha elegido entre los dispositivos encapsulados conocidos, por su nombre receptores infrarrojos, pero que se basan en los fotodiodos, la peculiaridad de estos dispositivos es que trabajan a una frecuencia determinada, normalmente 36 o 38 kHz. Estos dispositivos, son lo que tienen la mayoría de dispositivos para recibir señales infrarrojas, ya que al trabajar a una frecuencia determinada no se ven afectados por la luz artificial.

Tabla 2.7 Comparativa de receptores infrarrojos

Producto / REF	Frecuencia	Distancia máxima	Angulo de recepción	Precio
TSOP4P38	38 kHz	45 m	45º	0,50 €
TSOP38338	38 kHz	45 m	45º	0,56 €
TSOP38436	36 kHz	45 m	45º	0,56 €
Filtro previo de la búsqueda: Receptores Infrarrojos, Precio mínimo posible				

En este caso la elección viene condicionada por el menor precio posible, ya que estos sensores son capaces de trabajar con un espectro de infrarrojos variables. Una de las empresas líderes de estos dispositivos es la empresa VISHAY®, las diferencias entre los dispositivos son mínimas, por lo que para nuestras necesidades, la elección más económica es la mejor opción, en este caso se ha elegido el sensor TSOP4P38.

2.4.4 Acondicionamiento

El acondicionamiento tanto del emisor como del receptor de infrarrojos está relacionado, por eso se expondrá junto a este apartado. Lo que nos condiciona el diseño de este acondicionamiento es el receptor de infrarrojos. Para ello debemos mirar el datasheet del dispositivo cómo está condicionada la salida en función de la entrada. En la figura 2.6 podemos observar la salida del receptor que para un tren de pulsos recibidos a una frecuencia de 38 kHz,

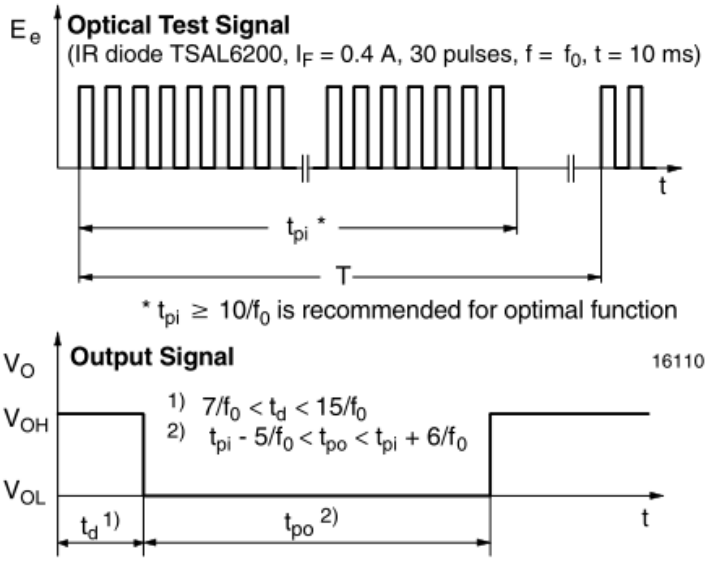


Fig. 2.6 Configuración de recepción de un TSOP4P38

Cómo vemos al recibir un tren de pulsos la señal del receptor pasará de HIGH a LOW. Este cambio de estado es la base de las comunicaciones por infrarrojos, ya que si somos capaces de detectar un cambio en la salida del receptor, sabremos si estamos recibiendo información, más adelante ya veremos cómo configuramos para saber si recibimos un 1 o un 0.

Uno podría pensar que esta configuración, sería más fácil si al recibir un tren de pulsos pasará de 0 a 1, y que nuestro sensor tiene esta configuración por ser de bajo coste, pues bien la gran parte de sensores infrarrojos funcionan de esta manera, TSOP382X, TSOP384X, TSOP392X, TSOP394, TSOP581X, TSOP583X, TSOP591X, TSOP593X, TSOP595X, TSOP348X son algunos de los ejemplos, por lo que saber acondicionar este tipo de sensores nos ayudará a tener herramientas para adaptarnos a la gran mayoría.

Una vez sabemos cómo se comporta nuestro receptor ya somos capaces de empezar a acondicionarlo. Primero el emisor, para acondicionar este sensor tenemos dos opciones o conectarlo directamente a través de un puerto digital, ya su salida no supera los 40 mA o través de un interruptor basado en transistor. El acondicionamiento del emisor de infrarrojos OPE5685 se puede ver en la figura 2.7. Si vemos el datasheet del componente, se puede ver que actúa como si fuera un diodo, su tensión de referencia es 1.5V.

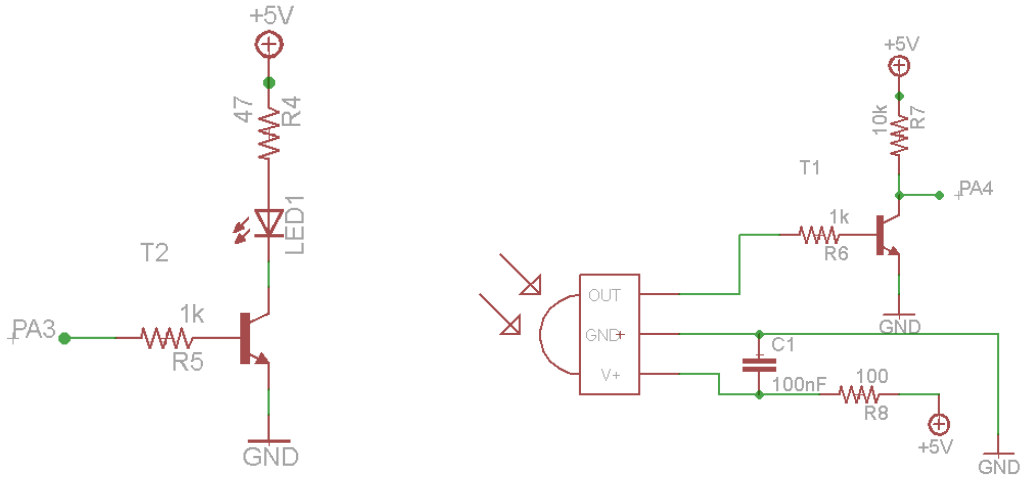


Fig. 2.7 Configuración de emisión y recepción de infrarrojos.

Por otra parte tenemos el receptor, si miramos el datasheet nos recomienda una configuración para alimentarlo y la seguiremos fielmente, el acondicionamiento propio se puede ver a la salida, con el transistor conseguimos transformar la señal que pasaba de 1 a 0 cuando recibe un tren de pulsos, a una que pase de 0 a 1, que nos facilitará mucho el trabajo a la hora de programar.

2.4.5 Calibración

Calibrar el sistema, emisor y receptor, es muy sencillo mediante un osciloscopio. Lo único que tenemos que hacer es enviar un tren de pulsos por el emisor, y ver que la salida de receptor reciba este tren de pulsos.

El proceso de calibración de este sistema se ha incluido en el anexo 3 de datos experimentales, al ser muy extensa su explicación y al contener un gran número de gráficos explicativos.

2.4.6 Resumen

Tabla 2.8. Tabla de los componentes del sensor de infrarrojos

Producto / REF/Link	Descripción	Cant.	Precio Unitario
OPE5685	Emisor de infrarrojos	2	0,680 €
TSOP4P38	Receptor de infrarrojos	2	0,50 €
BC548BZL1G	Transistor NPN	4	0,29 €
MCF 0.25W 1K	Resistencia ¼ W 1K	4	0,018 €
MCF 0.25W 10K	Resistencia ¼ W 10K	2	0,018 €
MCCFR0S2J0101A20	Resistencia ½ W 100	2	0,012 €
Coste completo del sistema de infrarrojos			3,652 €

2.5 Motores

Cómo se ha comentado anteriormente en la memoria, se ha utilizado un helicóptero RC como suministros de componentes, en concreto se ha aprovechado la batería y el bloque motor. El bloque motor de un helicóptero típico de 3 ejes, está formado por tres motores, dos motores para el rotor coaxial y un tercer motor para provocar el movimiento de cabeceo. Con estos tres motores seremos capaces de implementar movimiento en dos de los tres ejes, *pitch* y *yaw*. El movimiento que no seremos capaces de generar es el de *roll.*

Para hacer el acondicionamiento de los motores, usaremos un transistor con una entrada con modulación PWM. Una modulación PWM, *Pulse-Width Modulation,* es un modo de controlar la cantidad de energía enviada, mediante la variación del ciclo de trabajo de una onda periódica rectangular. Para que quede más claro, en vez de enviar una señal continua de 5 voltios, se envía una señal cuadrada variando la distancia entre los diferentes pulsos. Por ejemplo Arduino permite hacer una modulación PWM entre 0 y 255 valores. [29]

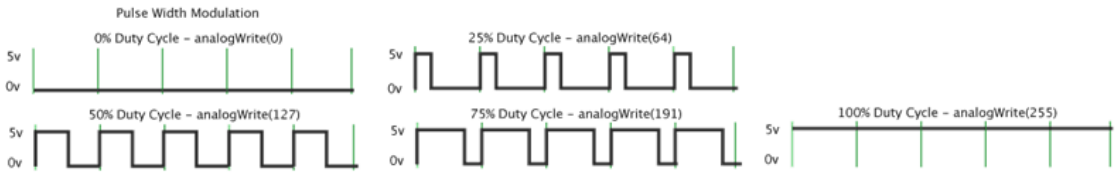


Fig. 2.8 PWM en Arduino [30]

El motivo para usar este tipo de modulación es que nos permitirá variar la velocidad del motor, ya que podremos controlar la cantidad de energía que recibe. En referencia al cambio de dirección de un motor, se suelen usar puentes en H, que no es más que una configuración de 4 transistores ordenados de una manera determinada para alimentar el motor. En nuestro caso cómo los motores de los rotores principales solo funcionan en un único sentido, no usaremos este tipo de sistemas.

En el acondicionamiento de motores, debemos de tener más precaución que en el desarrollo de otros acondicionamientos, básicamente por que trabajaremos con potencia, y debemos condicionar nuestros elementos a este parámetro. Por ejemplo si se usan resistencias en el colector del transistor, debemos de ser conscientes que estas resistencias tienen que poder disipar la potencia necesaria a la intensidad que circula y a su diferencia de potencial.

$$P = VI \tag{2.8}$$

Para adaptar los motores deberemos conocer la tensión y la intensidad que necesitan. En concreto los motores del helicóptero RC de pruebas se alimentan a 3.7V con una intensidad máxima de 750mA.

Para diseñar nuestro sistema de control de motores necesitaremos transistores que puedan tener una intensidad de colector de al menos 750mA, por esta razón los transistores BC548B utilizados hasta ahora no nos servirán para este propósito, estos tienen una I_c de 500mA. Como su sustituto se ha escogido el transistor BC337-25, para que pueda condicionar el sistema de motores.

Tabla 2.9 Transistor BC337-25

Producto / REF	TIPO	I_c	G,hfe	Precio
BC337-25ZL1G	NPN	800mA	160	0,81 €

Otro elemento muy importante a la hora de acondicionar motores, es su protección. A la hora de acondicionar motores debemos tener en cuenta que estos necesitan como mínimo de un diodo de protección para evitar las intensidades de retorno y proteger así el circuito. En la imagen 2.9 podemos ver el acondicionamiento.

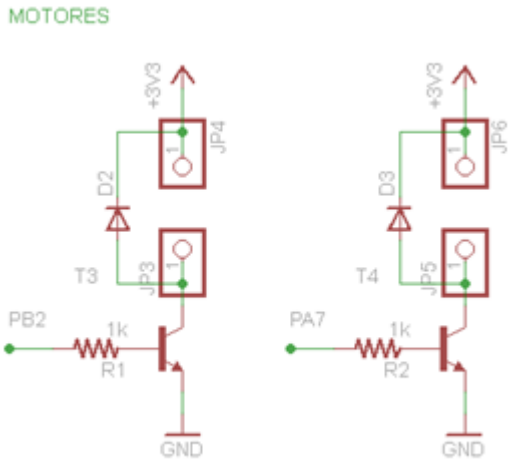


Fig. 2.9 Acondicionamiento de motores principales usando transistores

Con lo visto hasta ahora seriamos capaces de controlar la velocidad de un motor de continua en una única dirección. Si queremos controlar el motor en ambas direcciones deberemos usar un puente en H. Un puente en H es la unión de 4 transistores tal y como se muestra en la figura 2.10 y con este dispositivo seremos capaces de hacer circular la alimentación en dos sentidos, en una dirección y en su inversa.

En total para la actuación de los motores de un helicóptero, como el usado para realizar las pruebas necesitaríamos 6 transistores, 6 resistencias y 6 diodos de protección. Para ahorrar el espacio que ocuparían todos estos elementos en una placa de circuito impreso es más práctico utilizar un chip que tenga encapsulado toda el acondicionamiento para los motores, estos chips se conocen como drivers y un ejemplo muy popular es el drivers L293.

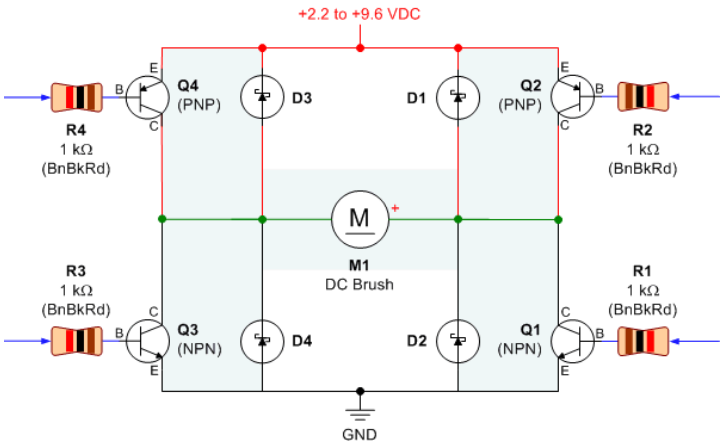


Fig. 2.10 Configuración típica de un puente en H [31]

Igualmente aunque la solución adoptada para este proyecto se utilizar un driver, se incluye la tabla con los componentes necesarios para realizar el acondicionamiento utilizando simplemente transistores NPN.

Tabla 2.10. Tabla de los componentes del acondicionamiento usando transistores (3 motores, 1 en puente en H).

Producto / REF/Link	Descripción	Cant.	Precio Unitario
BC337-25ZL1G	Transistor NPN 800mA	6	0,81 €
MCF 0.25W 1K	Resistencia ¼ W 1K	6	0,018 €
1N4148	Diodo de protección 150mA	6	0,087 €
Coste completo del acondicionamiento de 1 motor			5,490 €

2.5.1 Driver para motores

El driver elegido para la realización de este proyecto es L293N, cuyas características aparecen en la tabla 2.11

Tabla 2.11. Tabla de los componentes del acondicionamiento de motores

Producto / REF/Link	Descripción	Cant.	Precio Unitario
L293N	Motor Driver 1,2 A	1	3,13 €
1N4148	Diodo de protección 150mA	6	0,087 €
Coste completo del acondicionamiento de motores			3,652 €

Como podemos observar este tipo de acondicionamiento es más barato que usar transistores individuales. Igualmente todo dependerá de las necesidades del sistema, en nuestro sistema usar el driver es la solución más eficiente.

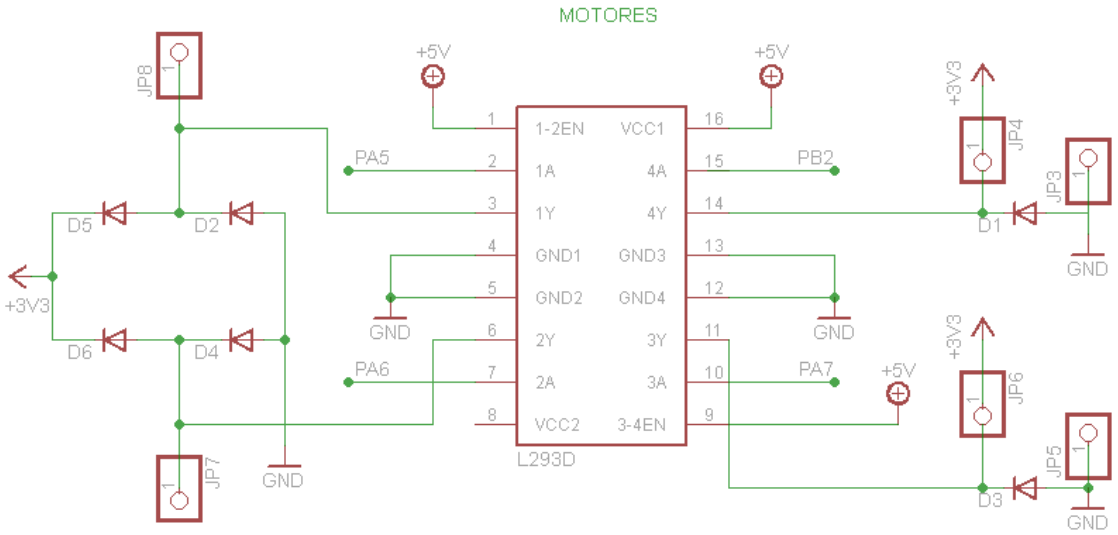


Fig. 2.11 Acondicionamiento de los actuadores del sistema

Un punto importante es que los drivers suelen trabajar con dos voltajes, el voltaje de alimentación de los motores, y el voltaje de nivel lógico. En la gran mayoría de sensores el nivel de alimentación de los motores debe ser mayor o igual que el voltaje de nivel lógico. Aunque en este proyecto se trabaje con una alimentación de 3.7V en la alimentación de los motores y 5V de nivel lógico el sistema funciona correctamente, ya que existe un margen de funcionamiento. Aunque es importante que para futuros proyectos se tenga en cuenta el nivel de voltaje de alimentación de los motores. Cómo alternativa siempre se pueden utilizar transistores si nuestro nivel de tensión en los motores es inferior a nuestro nivel lógico.

CAPÍTULO 3. COMUNICACIONES

3.1 Protocolo de comunicación entre estaciones

Las comunicaciones en este proyecto se basan en infrarrojos. Para la comunicación entre el Arduino y el ATtiny84, se ha decidido utilizar un protocolo de comunicación propio, a la medida de nuestras necesidades.

Básicamente lo que necesitamos es enviar información de un punto a otro. La primera pregunta que se nos presenta es: ¿Cuánta información debo transmitir? Básicamente lo que queremos es enviar información del ADC del ATiny84 a Arduino, y enviar información para el control de los motores entre el Arduino y el ATtiny84. Por lo que codificando 1024 valores tendremos suficiente. El primer paso será enviar información codificado en binario (pulsos de infrarrojos 0 y 1) para codificar 1024 valores.

Siguiente pregunta, como transmito 0 y 1 en infrarrojos, bien aquí el abanico es muy grande, se puede enviar pulsos solo para los 1, solo para los ceros, o nuestra elección, pulsos de diferentes duraciones para codificar 0 y 1. Los ceros serán 15 pulsos a 38kHz, mientras que los unos serán 30 pulsos a 38 kHz, cuando el receptor lea la duración del pulso sabrá si está recibiendo un uno o un cero. Ahora ya sabemos cuánto tenemos que enviar y cómo lo tenemos que enviar, pero necesitamos saber qué tipo de información estamos enviando, para esto añadimos 3 bits más al mensaje, esto nos permite separar ocho tipos de mensajes diferentes.

Cómo último debemos enviar una cabecera que diga al receptor que está recibiendo algo, esto no es estrictamente necesario, pero la verdad es que nos permitirá detectar errores en la transmisión, la cabecera serán otros 3 bits más en, este caso tres unos. Ya tenemos codificado nuestro protocolo de transmisión en total 16 bits. En la tabla 3.1 podemos ver un resumen de nuestro protocolo de transmisión.

Tabla 3.1 Ejemplo de mensaje de comunicación.

Cabecera			Mensaje			Información									
1	1	1	0	0	1	X	X	X	X	X	X	X	X	X	X

Tabla 3.2 Tipos de mensaje posible.

Mensaje			Tipo de información
0	0	1	Envío de información sensor temperatura
0	1	0	Envío de información sensor efecto hall
1	0	0	Envío de información sobre motor Principal 1
1	1	0	Envío de información sobre motor Principal 2
0	1	1	Envío de información sobre motor Cola dirección 1
1	0	1	Envío de información sobre motor Cola dirección 2

Para ver un poco más en detalle el protocolo utilizado, en la figura 3.1 se puede ver una captura de pantalla del osciloscopio donde se ve representado el bit 1 y el bit 0.

Fig. 3.1 Bit 1 (izquierda) y Bit 0 (derecha).

La duración del bit “1” es de 816 µs, mientras que en el bit “0” la duración es de 404 µs. La separación entre bits es constante y tiene una duración de 620 µs.

En la imagen 3.2 podemos ver la transmisión de una trama completa. El mensaje enviado corresponde a “1111000000000000”.

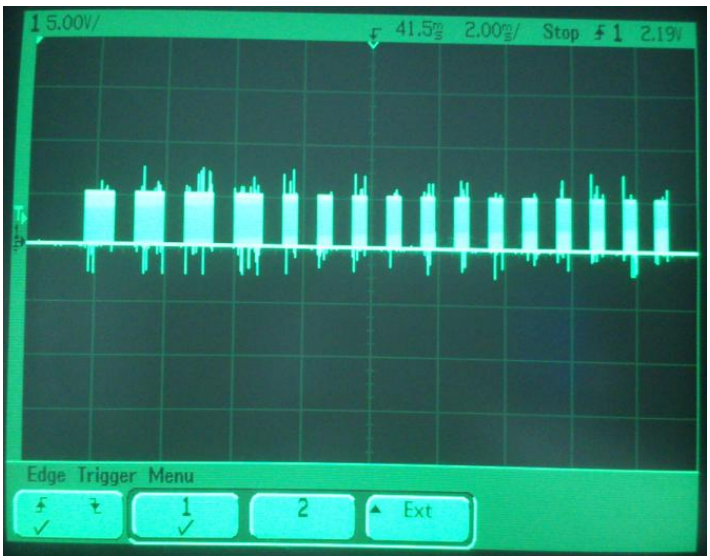


Fig. 3.2 Mensaje codificado en pulsos

3.2 Comunicación entre Arduino y el ordenador

En este apartado veremos muy básicamente cómo se comunica Arduino con el ordenador, este apartado no entra en detalles de comunicación, pero se incluye para dar una idea genérica de cómo es posible que Arduino realice una comunicación con un ordenador.

3.2.1 Conectando Arduino al ordenador

Arduino UNO, se comunica con el ordenador mediante USB. USB son las siglas de Universal Serial Bus, este es un protocolo estándar de comunicaciones vía serie. Básicamente y sin entrar en muchos detalles, un USB consta de 4 terminales, un pin de alimentación, un pin de masa y dos pines de comunicación, esta configuración se representa en la figura 3.3.

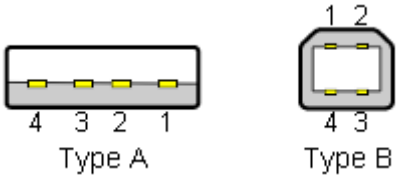


Fig. 3.3 Esquema de conexión USB [32]

Como vemos, existen dos canales de comunicación TX y RX, como su propio nombre indica uno es un canal de transmisión y otro de recepción. Existen chips que pueden ser conectados directamente a un cable USB, pero esto es solo una característica, y no es genérico para todos los microcontroladores. En nuestro caso necesitaremos un paso intermedio para conecta el chip a un cable USB, esto en la placa de Arduino UNO lo realiza el chip ATmega82U, que si tiene configuración USB, por lo que mediante una entrada TX y RX del USB, podemos convertir la señal mediante programación interna al protocolo SPI, (MISO, MOSI, SCK y RESET), para programar y comunicar nuestro ATmega328P-PU con el ordenador.

3.2.2 El protocolo RS232

Aunque es cierto que Arduino UNO recibe información a través del USB, en verdad lo que hace el driver o controlador de Arduino es emular un puerto COM, ¿Por qué? Por facilidad en el diseño del driver, por eso cuando elegimos donde está conectada nuestra placa Arduino seleccionamos un puerto COM más un número y no un USB más un número. Y ¿por qué no sale directamente una salida DB9, que es la salida RS232? Porque esta salida, ya no está presente en la mayoría de ordenadores, por eso seguramente, y esto es un comentario personal, se desarrolló una interfaz USB mediante un ATmega82U intermedio.

Si se quiere entender más acerca de este protocolo, se recomienda visitar la siguiente dirección: http://www.cursomicros.com/avr/usart/estandar-rs232.html

Lo que sí que necesitamos saber cuáles son las propiedades que representan un puerto RS232, para poder establecer correctamente la comunicación, si alguno de los parámetros que veremos a continuación no se adapta entre los sistemas de recepción y transmisión, existirán errores en el envío de los datos.

Los parámetros para establecer una comunicación serie RS232 son los siguientes: nombre del puerto, velocidad de transmisión, número de bits de datos, paridad, bits de parada y control de flujo.

El nombre del puerto, como su nombre indica, será elegir el puerto en el que está conectado nuestro dispositivo (COM1, COM2, COM3, etc.). La velocidad de transmisión, Baud Rate, se expresa en baudios. Un baudio es una unidad de velocidad que indica cuantos símbolos se han enviado por segundo, un símbolo es un cantidad determinada de bits de información, la cantidad de bits depende de la modulación, pero en RS232, un símbolo corresponde a un bit, por lo que Baut Rate coincide con el Bit Rate, velocidad de transmisión de bits. El número de bits de datos, es la cantidad de bits que se envían, normalmente se envía un byte, 8 bits. La paridad es un bit que se envía opcionalmente para detectar la detección de errores en la transmisión, normalmente la paridad es nula, lo que significa que no enviamos el bit. Los bits de parada son unos lógicos que indica que indican el final de la transmisión, normalmente este valor será igual a 1. Los bits de control de flujo son bits para el control de la información, este control se puede hacer mediante hardware o software pero lo más común es que no haya control de flujo.

Ahora que somos capaces, a grandes rasgos, de entender cómo se comunica Arduino con el ordenador, podemos ver una de las ventajas de trabajar de esta manera. La mayor ventaja que tenemos con este tipo de protocolo serie, es que existen librerías para casi todos los lenguajes de programación, por no decir todos, por lo que nos facilitará la tarea de exportar nuestra representación de datos a otro tipo de plataformas (matlab, visual C, python, etc.) y por eso solo se ha elegido una única plataforma (LabVIEW) para la representación de la información, ya que LabVIEW nos proporciona infinidad de funciones para el control de señales, todo de una manera visual, lo que es más intuitiva y fácil de usar que un lenguaje de programación convencional.

CAPÍTULO 4. PROGRAMACIÓN

La programación de los microcontroladores, tanto del ATtiny84 como del ATmega328, presente en el Arduino UNO; se ha realizado mediante el lenguaje de programación propio de Arduino. Los programa ATiny.ino y Arduino_UNO.ino se puede encontrar en el anexo 5, “Programación”.

En este apartado se entrará a detallar el código de una manera muy genérica para dar al lector una idea conceptual de la programación realizada en este proyecto, de manera que se entienda cómo están programados ambos microcontroladores.

```
void loop() {
enviarMensaje();
delay(50);
recibirMensaje();
delay(50);
}
```

Fig. 4.1 Esquema básico del programa de comunicación Arduino_UNO.ino

Como vemos en la figura 4.1 vemos el programa cargado en el microcontrolador del Arduino UNO, básicamente solo tenemos dos procedimientos: enviarMensaje y recibirMensaje.

El procedimiento enviar mensaje, lo que hace es leer información del puerto serie, dicha información es enviada por el LabVIEW a través de la interfaz de control. Arduino UNO lee el mensaje que esta codificado en una trama de 16 bits y envia a través de pin de transmisión un 1 o un 0 según corresponda. Como se ha visto en el apartado de sensores infarrojos, la información debe enviarse en pulsos a una frecuencia de 38 kHz, para ello simplemente se generan una señal cuadrada. Por ejemplo para enviar un pulso 1 se llama al procedimiento enviarPulsoAlto.

```
void enviarPulsoAlto() {

      for (int i=0; i<30;i++)
      {
        digitalWrite(TX, HIGH);
        delayMicroseconds(9);

        digitalWrite(TX, LOW);
        delayMicroseconds(9);
      }
    delayMicroseconds(600);            // Espera de 600 microsegundos.
  }
```

Fig. 4.2 Procedimiento enviarPulsoAlto en Arduino_UNO.ino y ATtiny.ino

En lo referente al procedimiento recibirMensaje, Arduino UNO utiliza una instrucción muy importante que se llama “pulseIN”, esta instrucción devuelve el tiempo de duración del pulso que ha recibido. Esta instrucción es muy importante por que es un limitador de velocidad del protocolo de transmisión,

ya que esta instrucción tiene un parametro de *timeout* que establece cuanto tiempo tiene que estar a la escucha para cada lectura del pulso, un timeout pequeño nos puede proporcionar una velocidad de transmisión muy elevada, pero por el contrario puede resultar dificil la sincronia entre la estación base y la estación remota.

```
void recibirMensaje(void)
{
      mensajeRecibido = 0;

      for(int i=0; i<16; i++)
      {
      duracion = pulseIn(RX, HIGH,1000);      // *

      if (duracion == 0)
      {
            break;
      }
      else if (duracion > 500)                // **
      {
            bitSet(mensajeRecibido, 15-i);
      }

   Serial.println(mensajeRecibido,BIN);       // ***
}

*  Límita la velocidad de transmisión
** Si el pulso dura más de 500 microsegundos, es un 1.
*** Imprime en binario el mensaje en el Serial.
```

Fig. 4.3 Procedimiento recibirMensaje de Arduino_UNO.ino

En lo referente a la programación del ATtiny84 es parecida, con la diferencia que la información que envia procede, no del LabVIEW, sino de los datos adquiridos por los sensores, Por ejemplo en la figura 4.4 vermos como se adquiere información del sensor y como se programa el envio al puerto de transmisión del emisor de infrarrojos.

```
infoTEMP    = analogRead(pinTemp);
cabecera    = 57344;  // "111" Cabecera
tipoMensaje = 1024;   // "001" Temperatura
informacion = word(infoTEMP);

mensaje = cabecera | tipoMensaje | informacion;
transmitirMensaje(mensaje);
```

Fig. 4.4 Transmitiendo información del sensor en ATtiny.ino

En el ATtiny84 al contrario que en Arduino UNO tenemos una procedimiento con un valor de entrada, que es el mensaje a transmistir. La forma de transmitir el mensaje es identica en los dos microcontroladores.

```
int value = 0;

    for(int i=15;i>=0;i--)
    {
      value = bitRead(mensaje_a_Transmitir,i);
      if (value == 0)
        enviarPulsoBajo();
      else
        enviarPulsoAlto();
    }
```

Fig. 4.5 Método utilizado para enviar información

Como vemos la programación en ambos microcontroladores es muy parecida, lo que es de gran ayuda, ya que podemos reutilizar partes del código.

Otra parte muy interesante es cómo codificar la información recibida en el ATtiny en una respuesta física en el actuador. Básicamente lo que hacemos es recibir el mensaje, igual que lo hacemos con el Arduino UNO, y una vez recibido el mensaje lo dividimos en tres partes: la cabecera, el tipo de mensaje y la información que lleva el mensaje. No entraremos en cómo partimos la cadena, ya que el código se puede consultar en los anexos, sino lo importante es cómo hacemos para variar la información de los actuadores, en nuestro caso nuestro motores. La figura 4.5 nos muestra cómo a través de un simple *switch* podemos variar la variable Z, que es la que controla el PWM del motor1.

```
...

 Z = int(informacion);
 elegir = int (tipoMensaje);
 cabeceraOK = int (cabecera);

    if (cabeceraOK == 7)
    {
      switch (elegir)
      {
        case 4:  // 100
             if (Z != motor1ANTERIOR)
             {
               analogWrite(motor1, Z);
               motor1ANTERIOR = Z;
             }
        break;
...
```

Fig. 4.6 Método utilizado para codificar información en el ATiny84

Para finalizar con este apartado, comentar que el programa se ha creado bajo la licencia GNU, para que puede ser utilizado y modificado libremente por cualquier persona, solo con la condición de utilizar en sus programas la misma licencia. Para no extenderse en este apartado se han puesto comentarios en todo el programa, para una fácil lectura del mismo y poder seguir el código con unos conocimiento muy básicos de programación.

CAPÍTULO 5. REPRESENTACIÓN DE DATOS

En esta última parte del proyecto, veremos cómo representar la información a través del programa Labview®. El programa Labview® es un programa basado en un lenguaje de programación gráfico que nos permite la manipulación de información a través de una interfaz visual muy sencilla pero a la vez muy potente, ya que nos permite realizar casi cualquier cosa.

Se ha elegido Labview® para realizar este proyecto, porqué se ha considerado una herramienta de desarrollo muy interesante además de intuitiva y fácil de aprender a utilizar, al menos a un nivel básico. También nos hemos decantado por este software porque permite la evaluación de sus productos sin ninguna limitación en un periodo de prueba por versión de treinta días.

Para ver cómo se ha realizado la interfaz gráfica en este proyecto, vamos a ver modularmente cada uno de los pasos realizados a través de Labview®.

5.1 Adquisición de datos del puerto Serie

El primer paso, es inicializar la comunicación a través del puerto Serie RS-232 y recibir la información desde el Arduino. Para esto necesitaremos además del programa, los drivers VISA, que son librerías para la comunicación en serie, estos drivers, son gratuitos y se pueden encontrar en la siguiente dirección, siendo la versión más reciente la 530:

http://joule.ni.com/nidu/cds/view/p/id/3823/lang/es

Una vez tengamos instalado el programa y los drivers, ya seremos capaces de conectarnos a nuestro puerto serie y adquirir información.

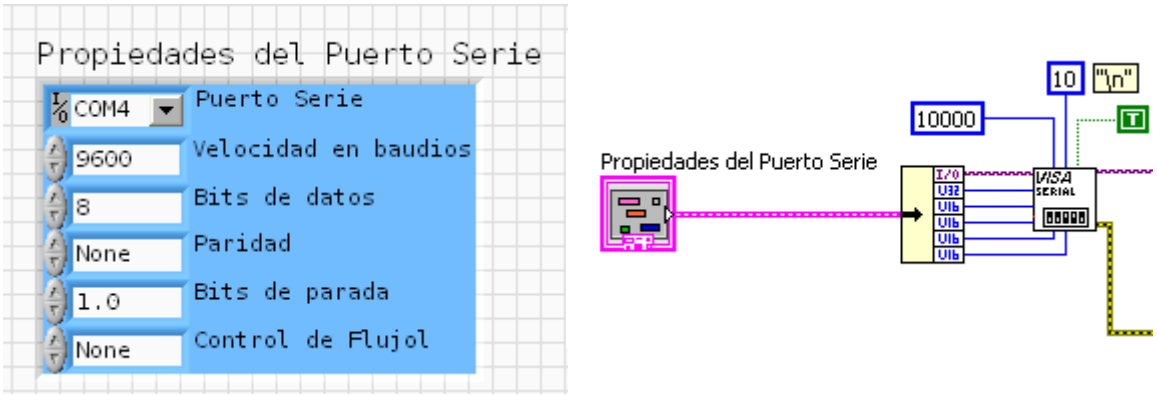


Fig. 5.1 Configuración del puerto serie RS-232 mediante VISA

En la imagen 5.1 podemos ver como se realiza, la configuración mediante el módulo “VISA SERIAL”, este módulo tiene una serie de entradas que se han de definir para establecer una comunicación entre el puerto RS-232 y Labview®, como por ejemplo, elección del puerto, velocidad del puerto, bits de datos, paridad, bits de parada y control de flujo. Además de estos parámetros que el

usuario podrá modificar el módulo “VISA SERIAL” admite dos entradas que serán muy útiles en nuestro proyecto, la entrada, carácter de final de línea, y que carácter queremos que sea el carácter de final de línea. En la figura 5.1 se puede ver como una entrada está configurada en “True”, esto quiere decir que el final de línea se dará cuando en el serial aparezca un salto de línea, esta configuración, es 10, en código ASCII.

5.2 DETECCIÓN DE ERRORES CONOCIDOS

Unos de los problemas que nos podemos encontrar si somos nuevos usuarios de Labview® es su configuración de errores, para facilitar el uso de la interfaz gráfica se ha considerado detallar los errores más comunes y los que han ido apareciendo cuando se realizaba la verificación del programa.

Básicamente se ha hecho una pequeño indicador, comparando los errores que van apareciendo en el programa, con los errores conocidos, por si al detectarse un error conocido, se imprime por pantalla un mensaje de error más detallado y más concreto, de lo que lo haría Labview®.Los errores se expresan en la tabla 5.1.

Tabla 5.1 Errores conocidos y tipificados en el programa.

Error	Explicación
-1073807343	Puerto COM, no válido.
-1073807202	Faltan los drivers VISA
-1073807246	Puerto COM en uso.

5.3 LEYENDO EL PUERTO SERIE

Al igual, que lo programado en los microcontroladores, en Labview también se entra en bucle, tal y como indica la figura 4.1 del apartado de programación, de transmisión, espera, recepción, espera, así indefinidamente, hasta que se le da la orden de terminar.

Para cambiar entre transmitir y recibir, se ha optado por utilizar una estructura case, con un indicador booleano controlado por un shift register, me explico, para cambiar entre transmitir y recibir, el programa tiene una señal de control true o false, que va cambiando cada iteración del bucle, por lo que si en la iteración N el valor es true, en la iteración N+1 el valor será false, así con esta señal de control tan simple, podemos intercambiar entre lectura y escritura. Se ha hecho de esta forma para si es necesario forzar al programa a solo transmitir o solo recibir información.

Otro punto a destacar, es que la lectura del puerto serie, se realiza en binario, para decodificar el código binario, lo primero que se hace es partir la cadena de números en segmentos que después pueden ser interpretados, por ejemplo si la cabecera es “111” sabemos que el mensaje es válido si el tipo de mensaje es “001” sabemos que se está recibiendo información de temperatura y por último la información del mensaje sí que debe ser pasada a un número decimal

para ser representada, esto se hace muy fácilmente mediante un bucle *for*, y nuevamente dividiendo la *string*, esta vez de uno en uno, y comparando cada valor con un 1, para que se introduzca por medio de un indexado a un convertidor de booleano a número decimal, me explico, lo que hace este parte del código es coger una cadena de números, y separarlos uno a uno, si el número es un 1 entonces tendrá un valor true, de lo contrario será false, el indexado lo que hace es guardarnos esa información hasta que acaba el bucle, por lo que cuando acabe, tenemos una serie de valores cierto y falso que pueden ser convertidos fácilmente a un número digital, mediante una módulo llamado “Boolean Array to number”, finalmente transformamos el número en decimal y ya lo podemos representar por pantalla.

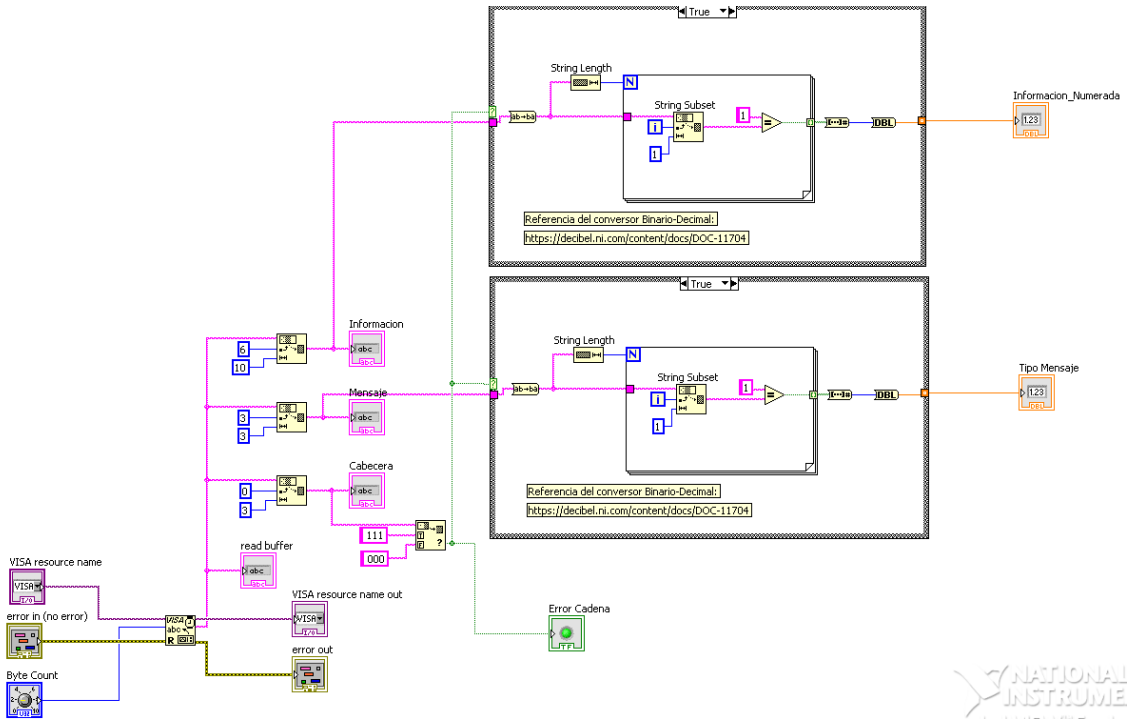


Fig. 5.2 Leyendo y codificando el mensaje recibido.

Como vemos, la manera de trabajar con Labview es muy sencilla, ya que existen infinidad de módulos, que permiten realizar las más diversas operaciones, e incluso si los módulos no existen los podemos crear nosotros mismos a partir de módulos más elementales.

5.4 ESCRIBIENDO EN EL PUERTO SERIE

A la hora de transmitir información lo que debemos hacer es escribir en el puerto serie, para eso y como ya hemos comentado anteriormente, con cada iteración variamos entre transmisión y recepción.

A la hora de escribir en el puerto serie lo tenemos más fácil aún que la lectura, pero tenemos que ser consciente de una serie de matices muy importantes. Primero, veamos la parte de código en la figura 5.3, cómo podemos observar la

escritura se realiza con muy pocos elementos, ya que el módulo "*Build text*" nos permite entrar una número y el mismo lo transforma a binario y además lo saca como cadena de caracteres.

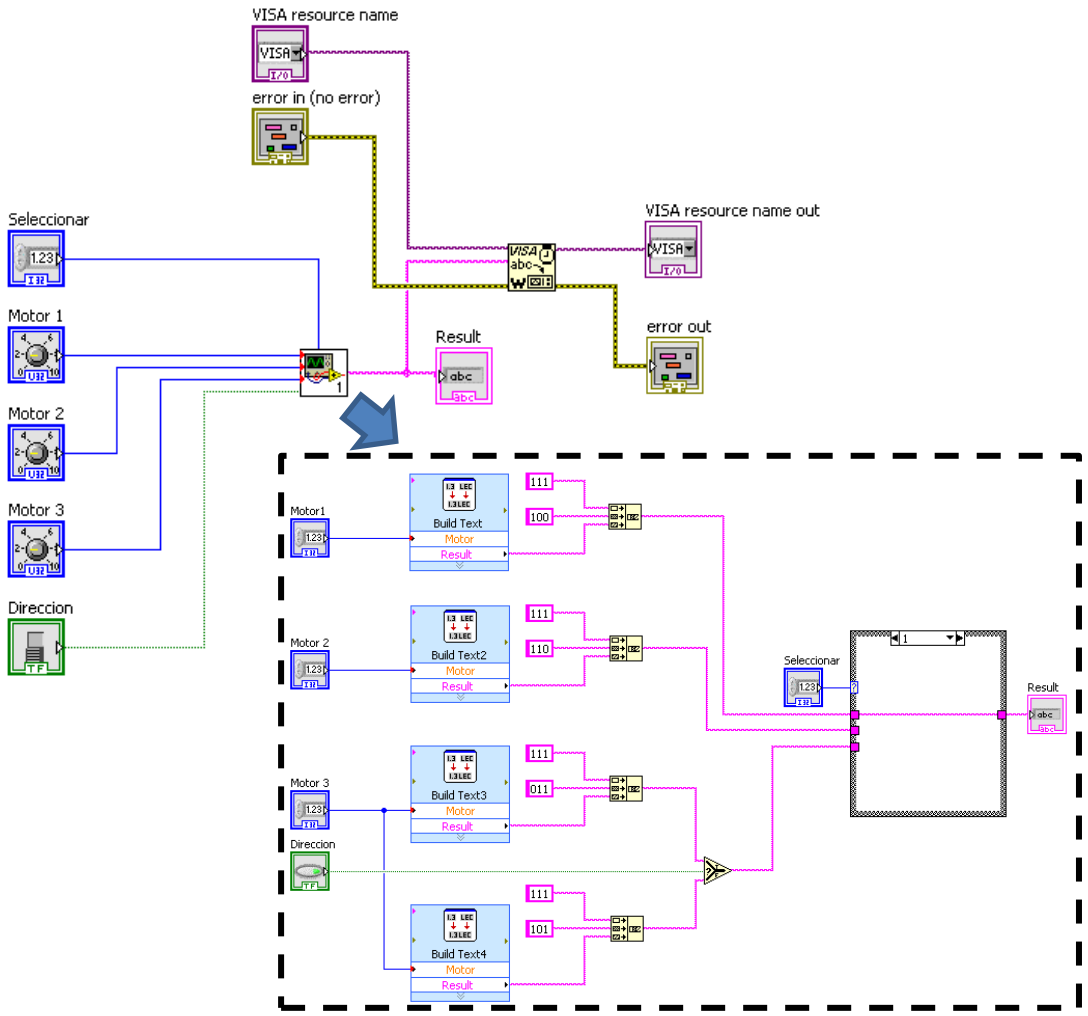


Fig. 5.3 Leyendo y codificando el mensaje recibido

Ahora lo único que tenemos que hacer es concatenar, la cabecera y el tipo de mensaje y ya estamos listos para enviar la información. En este caso en particular, como la información de los motores es solo PWM, solo podemos enviar números comprendidos entre el 0 y 255, para limitar el envío de números más grandes lo que se hace es limitar el control, en este caso podemos verlo a la izquierda de la imagen 5.3 que solo puede llegar desde el cero hasta el 255, con este sencillo sistema, nos ahorramos tener que comprobar la limitación del número.

5.5 INTERFAZ

La representación de la interfaz y una explicación más extendida del sistema se puede encontrar en el Anexo 6 " Esquemas de LabVIEW". Como visión de conjunto se puede encontrar una explicación del sistema en el anexo 7.

CAPÍTULO 6. CONCLUSIONES

Se han conseguido alcanzar los objetivos propuestos al inicio del proyecto. El sistema desarrollado cumple las expectativas iniciales, de ser un sistema de adquisición y control que trabaje de forma inalámbrica, económico y donde se ha tratado de facilitar su comprensión para que puede ser de utilidad como sistema didáctico que permita el aprendizaje básico de los sistemas de adquisición y actuación.

Se han adaptado con éxito los sensores de temperatura y posición, como parte del sistema de adquisición de datos. Además se han acondicionado los motores como parte del sistema de actuación del sistema.

En lo referente a los microcontroladores, pieza central del proyecto, se ha aprendido desde programar los fuses internos, hasta la realización de programas en Arduino para controlar el sistema inalámbrico.

En el apartado de comunicaciones, el protocolo desarrollado, aunque básico cumple correctamente su cometido. Además el protocolo creado es muy sencillo de implementar, y se puede adaptar a diferentes necesidades.

Cómo último punto, destacar el potencial que tienen las herramientas de Arduino y LabVIEW para la realización de todo tipo de proyectos, ya que ambos unidos crean un entorno de desarrollo sencillo y eficiente, además de ser plataforma con una comunidad que ofrece soporte de muy buena calidad.

Para líneas futuras de desarrollo se deja la mejora de los sistemas adquisición, añadiendo acelerómetros y giroscopios, la miniaturización de los módulos a elementos SMD, que aumentarían notablemente la posibilidad de embarcar el sistema en un MAV, y poder ampliar el sistema de comunicación inalámbrico a radio-frecuencia para aumentar el alcance de transmisión.

En lo que se refiere a la ambientalización, este proyecto se enmarca a favor de un desarrollo sostenible, utilizando elementos que cumplen la normativa RoHS, *Restriction of Hazardous Substance,* normativa europea recogida en la Directiva 2002/96/CE que tiene como objetivo la restricción de sustancias peligrosas en equipos electrónicos.[33]

BIBLIOGRAFÍA Y REFERENCIAS

[1] Clasificación de vehículos aéreos no tripulados:
www.gmat.unsw.edu.au/currentstudents/ug/projects/Salameh/Physical%20Characteristics.htm

[2] Fig. 0.1 Ejemplos de MAV's:
chivethebrigade.files.wordpress.com/2013/02/black-hornet-uav-500-6.jpg
http://people.csail.mit.edu/albert/images/kinectquad-200.jpg

[3] Especificaciones Arduino UNO REV 3:
http://arduino.cc/en/Main/arduinoBoardUno

[4] Fig. 1.1 Plataforma Arduino:
http://hacromatic.com/images/large/arduino_uno_r3_1_LRG.jpg

[5] Fig. 1.2 Pines de Arduino UNO:
http://makezineblog.files.wordpress.com/2011/12/arduinounor3_lrg.jpg

[6] Programa de ejemplo Blink, con anotaciones propias en castellano.

[7] Precio de Arduino Duemilanove en http://store.arduino.cc/eu. Fecha 15/04/2013.

[8] Precio aproximado ATmega328P-PU en http://es.farnell.com, precio real 3,86€ (Sin IVA). Fecha 15/04/2103.

[9] Programación en ATtiny de GitHub:
https://github.com/damellis/attiny/issues

[10] Programación en ATtiny del MIT:
http://hlt.media.mit.edu/?p=1695

[11] Programación en ATtiny de Arduino-Tiny:
http://code.google.com/p/arduino-tiny/

[12] Documentación de microcontroladores tinyAVR®.
http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx

[13] Fig. 1.4 Microcontroladores ATtiny84 y ATtiny85:
http://hlt.media.mit.edu/wp-content/uploads/2011/10/ATtiny44-84.png
http://hlt.media.mit.edu/wp-content/uploads/2011/10/ATtiny45-85.png

[14] Especificaciones técnicas ATtiny85 & ATiny84:
http://www.atmel.com/devices/ATTINY85.aspx
http://www.atmel.com/devices/attiny84.aspx

Especificaciones técnicas ATtiny84:
http://www.atmel.com/devices/attiny84.aspx
http://www.atmel.com/Images/doc8006.pdf

[15] [Excelente manual sobre microcontroladores AVR: http://www.cursomicros.com/avr/programadores/interfaces-de-programacion.html

[16] Fig. 1.6 Arduino como ISP conectado a ATtiny84
Esquema realizado con fritzing, http://fritzing.org/

[17] Imagen propia, basada en la información de:
"tiny\cores\tiny\core_pins.h" de la librería:
http://code.google.com/p/arduino-tiny/

[18] Pallás Areny, R., "Introducción a la adquisición y distribución de señales", Cap. 1 en *Adquisición y distribución de señales*, MARCOMBO, pp. 1-26, Barcelona (1993).

[19] Fig. 1.10 Conversión Analógica a Digital con ADC de 3 bits:
www.prz.rzeszow.pl/kpe/materialy/astadler/DAQWWW/LabVIEW/DAQ/Elementytoruakwizycji/Image3.jpg

[20] Pallás Areny, R., "Sensores y actuadores", Cap. 2 en *Adquisición y distribución de señales*, MARCOMBO, pp. 27-84, Barcelona (1993).

[21] Datasheet del MAX756:
http://pdfserv.maximintegrated.com/en/ds/MAX756-MAX757.pdf

Datasheet del LT1303:
http://cds.linear.com/docs/en/datasheet/lt1303.pdf

[22] Casanella, R. "Especificaciones de un sistema de medida / actuación.", Apuntes de la asignatura de Diteva.

[23] Casas, O., Apuntes de la asignatura de Aviónica, UPC (2012).

[24] Fig. 2.1 Simbología de los sensores de temperatura resistivos:
http://tecnoveron.blogspot.com.es/2011/02/termistores.html

[25] Pallás Areny, R., "Sensores resistivos", Cap. 2 en *Sensores y acondicionamiento de señal*, MARCOMBO, pp. 54-97, Barcelona (2007).

[26] Teoría de infrarrojos:
https://en.wikipedia.org/wiki/Infrared

[27] Teoría del espectro visible:
http://es.wikipedia.org/wiki/Espectro_visible

[28] Definición de fotodiodo:
http://es.wikipedia.org/wiki/Fotodiodo

[29] Definición de modulación PWM:
http://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

[30] Fig. 2.9 PWM en Arduino:
http://arduino.cc/es/Tutorial/PWM

[31] Fig. 2.10 Configuración típica de un puente en H:
http://www.robotroom.com/BipolarHBridge/BipolarHBridgeSchematic.gif

[32] Fig. 3.3 Esquema de conexión USB:
http://en.wikipedia.org/wiki/File:USB_Std_A.png
http://en.wikipedia.org/wiki/File:USB_Std_B.png

[33] Definición de RoHS:
http://es.wikipedia.org/wiki/RoHS

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTULO DEL TFC: Diseño e implementación de un sistema de adquisición y actuación inalámbrico para vehículos aéreos no tripulados.

TITULACIÓN: Ingeniería Técnica Aeronáutica, especialidad Aeronavegación

AUTOR: Jorge Polo Alonso

DIRECTOR: Ramón Casanella Alonso

FECHA: 3 de junio de 2013

ÍNDICE

INTRODUCCIÓN

Los anexos de este proyecto, se han utilizado como extensión de la memoria y como soporte a conceptos necesarios para entender el funcionamiento del sistema.

Adicionalmente los anexos, se han usado como ampliaciones a los objetivos iniciales del proyecto.

ANEXO 1. ACONDICIONANDO EL ATTINY84

En este primer anexo veremos cómo podemos adaptar el ATtiny84 a nuestras necesidades.

1.1 Inicializando los fuses del microcontrolador ATtiny84

Los microcontroladores cuando salen de fábrica vienen con una serie de características definidas por sus fuses. Los fuses son registros del microcontrolador que sirven para establecer una serie de características, modificando los fuses podemos forzar al microcontrolador a usar un cristal externo, o por el contrario usar la frecuencia de trabajo interna, etc… [1]

Cada microcontrolador, tiene asociado una serie de fuses, que pueden ser modificados, y para facilitarnos la tarea de definirlos, podemos consultar páginas como: http://www.engbedded.com/fusecalc/. Esta página nos proporciona una herramienta muy útil a la hora de programar los fuses, ya que da soporte a una gran cantidad de microcontroladores de la marca Atmel®.

En este proyecto necesitamos establecer los fuses para que nuestro microcontrolador trabaje con un cristal externo a una velocidad de 16 MHz, si no hiciéramos esto, el microcontrolador de fábrica trabajaría con su frecuencia interna que es de 1 o 8 MHz.

Para establecer los fuses, necesarios podemos consultar el archivo “boards.txt” que podemos encontrar en la carpeta “\tiny” del archivo que nos hemos descargado en el apartado anterior en la página de Arduino-Tiny.

El archivo “boards.txt” establece los fuses necesarios para cada frecuencia de uso, en nuestro caso usaremos el archivo de 16MHz, una imagen de este fragmento de archivo se encuentra en la imagen 1.1, en este archivo podemos observar los parámetros a los cuales tenemos que programar los registros fuses de nuestro microcontrolador, resaltados en color verde. Usando la página fusecalc, podemos ver la configuración que nos aconsejan utilizar para utilizar nuestro microcontrolador con un cristal externo a 16MHz, en la imagen 1.2 podemos ver la configuración de cada uno de los registros de los fuses.

```
attiny84at16.name=ATtiny84 @ 16 MHz (external crystal; 4.3 V BOD)
attiny84at16.upload.using=arduino:arduinoisp
attiny84at16.upload.maximum_size=8192
attiny84at16.bootloader.low_fuses=0xFF
attiny84at16.bootloader.high_fuses=0xD4
attiny84at16.bootloader.extended_fuses=0xFF
attiny84at16.bootloader.path=empty
attiny84at16.bootloader.file=empty84at16.hex
attiny84at16.build.mcu=attiny84
attiny84at16.build.f_cpu=16000000L
attiny84at16.build.core=tiny
```

Fig. 1.1 Parte del archivo boards.txt de Arduino-Tiny

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.

Note: ☐ means unprogrammed (1); ☑ means programmed (0).

Bit	Low	High	Extended
7	☐ CKDIV8 Divide clock by 8	☐ RSTDISBL External Reset disable	
6	☐ CKOUT Clock Output Enable	☐ DWEN DebugWIRE Enable	
5	☐ SUT1 Select start-up time	☑ SPIEN Enable Serial Program and Data Downloading	
4	☐ SUT0 Select start-up time	☐ WDTON Watchdog Timer always on	
3	☐ CKSEL3 Select Clock source	☑ EESAVE EEPROM memory is preserved through the Chip Erase	
2	☐ CKSEL2 Select Clock source	☐ BODLEVEL2 Brown-out Detector trigger level	
1	☐ CKSEL1 Select Clock source	☑ BODLEVEL1 Brown-out Detector trigger level	
0	☐ CKSEL0 Select Clock source	☑ BODLEVEL0 Brown-out Detector trigger level	☐ SELFPRGEN Self-Programming Enable

Fig. 1.2 Fuses para utilizar un cristal externo de 16MHz. [2]

Para realizar la programación de fuses, se tiene que hacer antes, de cargar el programa, y se necesitará un programador AVR. Un programador AVR es un hardware que permite programar los chips mediante un protocolo de comunicación SPI, y lo más importante es que nos permitirá borrar el chip, y establecer los nuevos fuses, Arduino solo permite hacer esto en línea de comando, y usando un programador es mucho más fácil y además con una interfaz visual. Este es el único paso en el que no utilizaremos Arduino como programador ISP, a partir de este punto el programador AVR ya no será necesario, y solo con Arduino UNO podremos programar nuestro microcontrolador ATtiny.

Antes de realizar la programación de los fuses, es necesario aprender un poco sobre el protocolo SPI y como se programan microcontroladores con programadores ISP.

1.1.1 Programando los fuses del ATtiny

Para entender un poco mejor la teoría del apartado anterior, veamos una imagen de como conectar nuestro programador AVR a nuestro ATtiny84. La elección del programador es a elección personal, como para este proyecto solo se necesita el programador para programar los fuses, recomiendo comprar uno lo más barato posible, el usado aquí cuesta 7,72 euros en la página: http://www.miniinthebox.com/es/ REF: “usb isp usbasp programador para atmel avr”, aunque si se está interesado en la programación de microcontroladores Atmel®, recomiendo programadores compatibles con AVR Studio, como por ejemplo USBTiny http://www.ladyada.net/make/usbtinyisp/ o AVRISP http://www.atmel.com/tools/AVRISPMKII.aspx.

En nuestro caso el programador tiene una compatibilidad muy limitada de programas, y solo se puede utilizar con el programa PROGISP v1.72 aunque para nuestro cometido es más que suficiente. Nuestro programador AVR tiene una salida con un conector de 10 pines como el que se muestra en la figura 1.3.

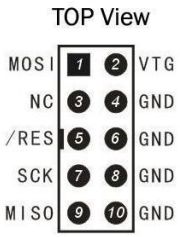


Fig. 1.3 Conector de salida del programador AVR. [3]

Ahora solo falta conectar cada pin a su correspondiente en el programador, en la figura 1.4 se puede ver un esquemático de la conexión, el esquema se ha desarrollado con la herramienta gráfica fritzing, http://fritzing.org/.

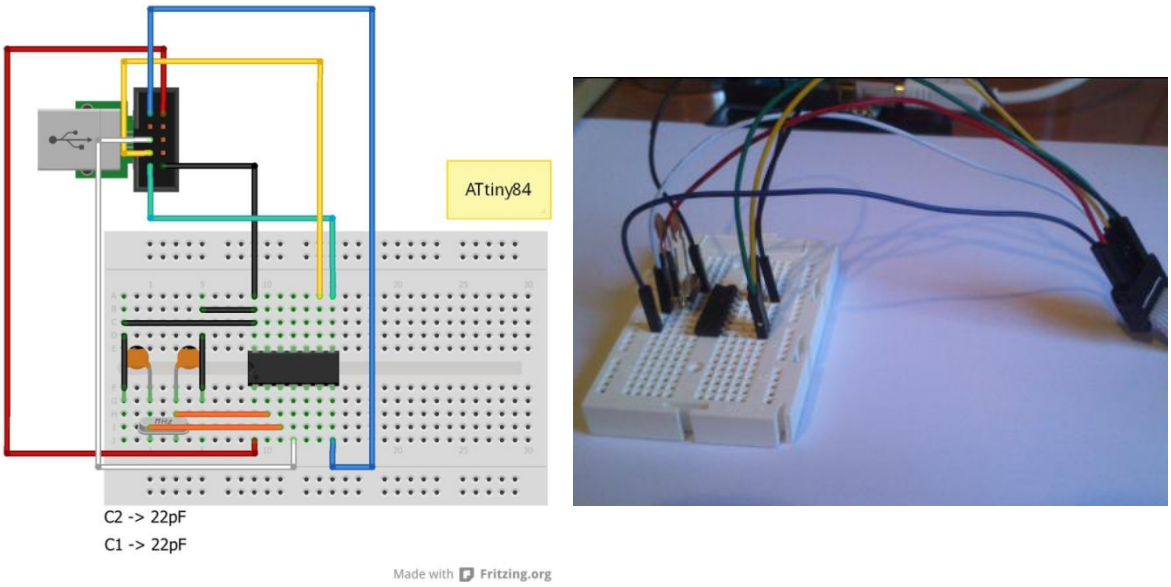


Fig. 1.4 Esquema para programar fuses del ATtiny. [4]

Ahora solo hace falta conectar el programador al ordenador, y ejecutar el programa PROGISP. El reconocimiento del USB, es automático por lo que no hace faltan drivers.

Lo primero que nos aparece es una interfaz como la que sale en la figura 1.5, seguidamente lo que tenemos que hacer es seleccionar el chip en la pestaña “Select chip”, en nuestro caso seleccionaremos el ATtiny84. Seguidamente y seleccionando las pestañas marcadas en la imagen procederemos al borrado del chip, para ello lo único que tenemos que hacer es clicar donde pone “Erase”. A continuación se procede al borrado del chip, si todo ha salido bien, saldrá un mensaje de control “Chip Erase succesfully”, si no es el caso,

seguramente los pines SPI no están bien conectados, o no hemos elegido el chip correcto.

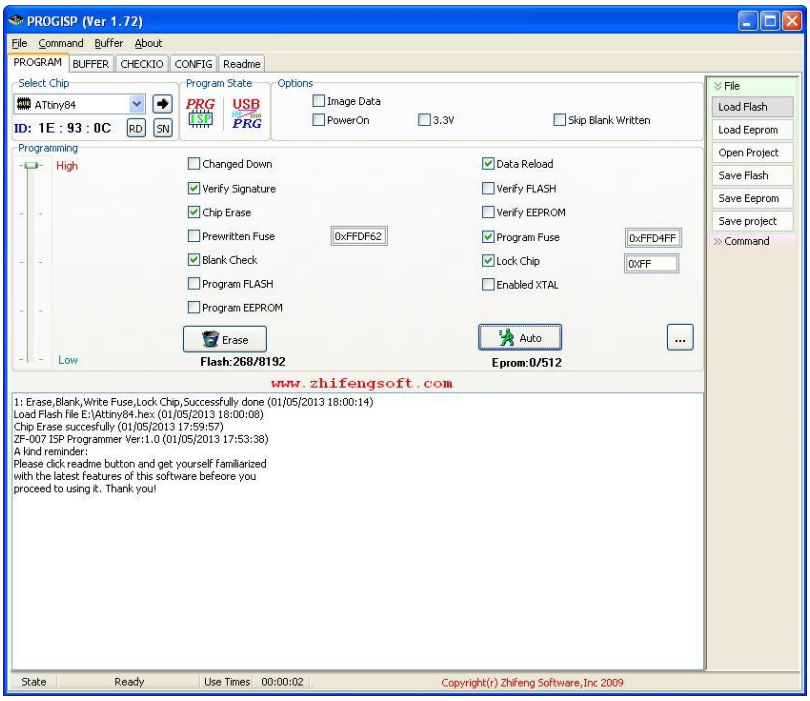


Fig. 1.5 Pantalla del programa PROISP v 1.72. [5]

Lo que haremos a continuación es seleccionar el Bootloader del ATiny, este lo podremos encontrar en la carpeta: \tiny\bootloaders\empty\empty84at16.hex para ello seleccionaremos “Load Flash”. Lo siguiente es seleccionar los fuses adecuados, para ello lo que haremos es clicar sobre la ventana, al lado de Program Fuse, y saldrá una ventana como la figura 1.6.

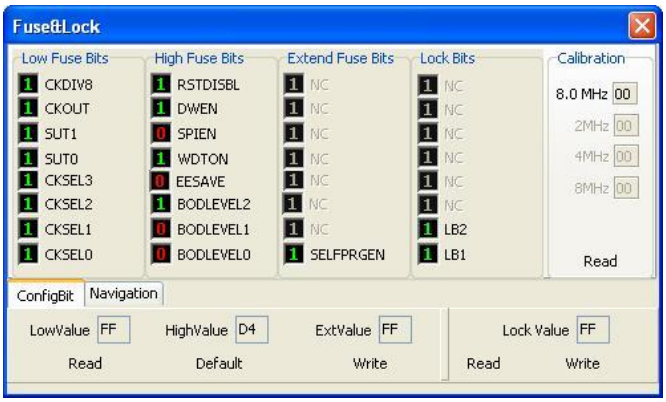


Fig. 1.6 Pantalla del programa PROISP v 1.72. [5]

A continuación seleccionaremos los valores FF, D4 y FF tal y como muestra la figura 1.6. Una vez hecho esto lo único que nos queda por hacer es clicar sobre el icono “Auto”, y después de un proceso de carga, nos aparecerá el mensaje “Erase, Blank, Write Fuse, Lock Chip, Succesfully done” y “Load Flash file” con el archivo que hayamos escogido. Y listo ya tenemos nuestro ATtiny listo para ser programado en Arduino a 16 MHz.

ANEXO 2. ELEMENTOS BÁSICOS

En este apartado veremos la teoría básica para ser capaces posteriormente de realizar los acondicionamientos de los sensores. Los elementos básicos para acondicionar los sensores en este proyecto serán: el amplificador operacional y los transistores.

Cómo uno de los objetivos es dar herramientas al lector para realizar sus propios diseños, no entraremos excesivamente en teoría de circuitos, aunque en cualquier libro de electrónica básica se puede aprender el análisis numérico de estos dos dispositivos.

Tengo que aclarar que en este proyecto trabajaremos con transistores NPN, que son una clase de transistores BJT, se ha tomado la decisión de trabajar con estos transistores porque son con los que se ha trabajado a lo largo de la carrera, y son muy fáciles de implementar, aunque se es consciente que los transistores FET presentan muchas ventajas sobre los BJT, menos ruido, una impedancia de entrada más alta, menor consumo, conmutación más rápida, etc.

2.1 El amplificador operacional

El amplificador operacional es un elemento electrónico diseñado en principio para realizar operaciones aritméticas, con este elemento se pueden realizar operaciones tales como: sumas, restas, derivación e integración de señales. En nuestro caso lo que nos interesa es la capacidad del amplificador operacional de amplificar un señal, multiplicando la señal de entrada por un factor X, que se denomina ganancia.

Cómo ya hemos comentado, no entraremos en el análisis numérico, pero si que daremos las ecuaciones de salida necesaria para usar de manera rápida e intuitiva esta poderosa herramienta. El símbolo de un amplificador operacional y su ecuación de salida cuando trabaja en modo de amplificador no inversor se puede ver en la figura 2.1, este modo de trabajo es el típico uso para un amplificador operacional, pero existen muchos otros, entre ellos se pueden destacar: el seguidor de tensión, el amplificador inversor, el amplificador con entrada diferencial, etc.

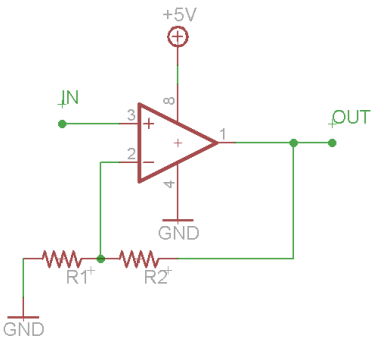


Fig. 2.1 Amplificador no inversor

La ecuación que rige este tipo de diseños es la 2.1.

$$V_{out} = V_{in}\mathrm{G} = V_{in}(^{R2}/_{R1} + 1) \quad \textbf{(2.1)}$$

Como vemos con este sencillo elemento, podremos controlar la amplificación variando sólo los valores R2 y R1, siempre teniendo en cuenta unos límites marcados por el fabricante. También es importante destacar que nosotros siempre trabajaremos con alimentación simple o unipolar, o lo que es lo mismo entre GND y 5V. Otra forma de trabajar con los amplificadores operacionales es mediante alimentación simétrica, por ejemplo de -5V a 5V, pero no es nuestro caso. Existen muchos tipos de amplificadores operacionales en el mercado, los más recomendado para aplicaciones con un cierto grado de precisión, son amplificadores que tienen la característica RAIL to RAIL, esto significa que el operacional es capaz de darnos a la salida tensiones muy cercanas a la tensión a la que alimentemos el amplificador, por lo que aprovecharemos mejor el rango de salida. A modo de orientación, un ejemplo de amplificador operacional de alimentación simple, bajo coste, RAIL to RAIL y bajo ruido es el TLC2272.

2.2 El transistor

El transistor es el elemento fundamental en la electrónica digital, su función principal es trabajar como un interruptor binario, pudiéndose crear con él todas las puertas lógicas, por lo que con un transistor se puede hacer casi cualquier cosa, básicamente se puede decir que el transistor es la base de nuestra tecnología actual.

Ahora bien, ¿Qué es? y ¿Cómo funciona? Un transistor es básicamente la unión de tres elementos semiconductores, y es capaz de permitir la circulación de corriente entre dos de las capas en función de la tercera, con eso de momento nos basta. Existen dos tipos de transistores, los BJT y los FET, básicamente se diferencian en cómo están organizadas las capas semiconductoras y en cómo trabajan, los BJT trabajan con intensidad en la entrada, mientras que los FET trabajan con diferencia de potencial

El transistor NPN, tienen tres pines tal y cómo se indica en la figura 2.2, la base (B), el colector (C) y el emisor (E).

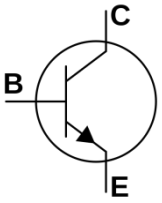


Fig. 2.2 Simbología de un transistor NPN [6]

Cómo en capítulos anteriores, no entraremos en detalles sobre el análisis de estos dispositivos, sino cómo configurarlos para que nos sirvan de herramienta, en el caso del transistor, el uso que vamos a darle es de un interruptor, de esta manera con una pequeña intensidad en la base podremos conseguir una mayor intensidad que circule a través del colector. Básicamente el transistor tiene tres regímenes de trabajo: activo, saturación y corte, aunque a nosotros solo trabajaremos en saturación y corte, ON y OFF.

Se trabajará en corte cuando V_{BE} sea menor o igual a 0.7, este valor depende del tipo de transistor, este valor nos lo da el datasheet del fabricante, en el caso del transistor BC548B que es uno de los utilizados en este proyecto este valor es igual a 0.7. Cuando VBE sea menor que 0.7 la intensidades que circulan por todas las patillas del transistor serán cero.

$$V_{BE} \leq 0.7V \rightarrow I_b = I_c = I_e = 0 \quad \textbf{(2.2)}$$

En régimen de saturación tenemos un valor V_{CE} que también nos proporciona el fabricante, en nuestro caso 0.25V, en esta caso la corriente que circulará por el colector será menor o igual a β veces I_B, siendo β la ganancia proporcionada por nuestro transistor, en nuestro caso 450.

$$V_{CE} = 0.25V \rightarrow I_C \leq \beta\, I_B \quad \textbf{(2.3)}$$

Ahora si despejamos las ecuaciones de los dos casos, veremos lo fácil que es trabajar entre corte y saturación solo eligiendo R_B y R_c apropiadas. Primero suponemos que estamos en zona de saturación y VCC es 5V, por lo que según la ley de Ohm, tenemos:

$$I = {}^{V}/_{R} \rightarrow I_c = {}^{(5V - 0.25V)}/_{R_c} \quad \textbf{(2.4)}$$

Si escogemos el valor de la intensidad que necesita nuestro componente por ejemplo el máximo que nos proporciona Arduino son 50 mA en su pin de salida a 5V, entonces solo tendríamos una incógnita, R_c, al despejarla tendríamos que R_c debería ser 95 Ω. Sabiendo la I_c también podemos despejar I_B de la ecuación 2.3 y nos queda la siguiente inecuación:

$$ {}^{I_C}/_{\beta} \leq I_B \quad \textbf{(2.5)}$$

I_B debe ser mayor que I_C entre β, por lo que si despejamos I_B tiene que ser mayor o igual a 111,11 µA. Si cogemos un margen de seguridad suficientemente alto podemos coger I_B como 4 mA, si despejamos R_B en la ecuación 2.6 ya tendremos resuelto nuestro interruptor.

$$I_B = {}^{5V - 0.7V}/_{R_B} \quad \textbf{(2.6)}$$

Lo que nos da que R_B será igual a 1075 Ω. Para usar valores comerciales siempre usaremos resistencias de 1KΩ en la base de los transistores, como el margen es grande, adaptaremos R_C a las necesidades de cada momento.

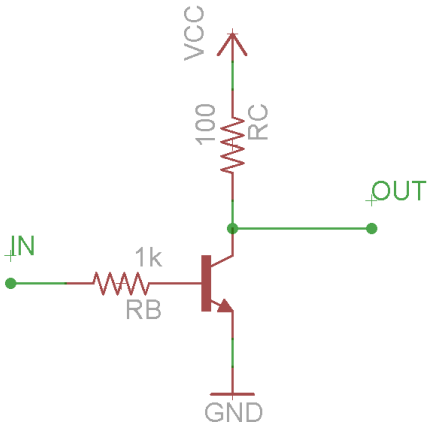


Fig. 2.3 Modelo de un transistor NPN trabajando como interruptor

2.3 Filtros

Los filtros son sistemas formados por resistencias y condensadores o resistencias y bobinas. Estos sistemas tienen como objetivo dejar pasar o detener, depende el tipo de filtro, un determinado ancho de banda.

En este proyecto nos centraremos en un tipo concreto de filtro, el filtro pasa-bajas. Este tipo de filtro tiene la misión principal de dejar pasar las frecuencias por debajo de una frecuencia umbral f_0 y atenuar o suprimir las frecuencias por encima de este umbral.

Una imagen de un típico filtro pasa-bajas se representa en la figura 2.4.

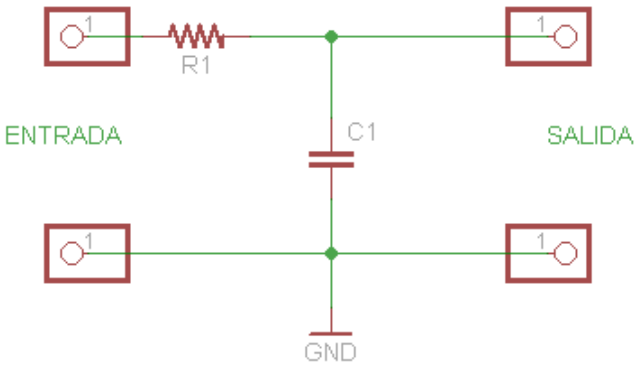


Fig. 2.4 Filtro pasa-bajas, en una configuración típica.

Para diseñar un filtro pasa-bajas tenemos a nuestra disposición la fórmula 2.7, si asignamos la frecuencia, tendremos dos variables para ajustar la ecuación.

$$f = {}^{1}/_{2\pi RC} \qquad \textbf{(2.7)}$$

Una configuración típica es utilizar un filtro pasa baja-bajas de 1 Hz, para ello podemos establecer los valores siguientes: R = 1MΩ y C=100nF.

ANEXO 3. RESULTADOS EXPERIMENTALES

En este apartado del proyecto veremos algunos datos experimentales obtenidos durante la realización de los módulos de diseño.

Este apartado ayudará a comprender mejor el trabajo realizado, además de aportar una serie de datos, que pueden ser utilizados como ejemplo y/o como guía adicional al texto de la memoria.

3.1 Unidad de Procesamiento

Vamos a resumir un poco los puntos de la memoria como introducción, la unidad de procesamiento se basa en dos elementos, la estación de base y la estación remota. La estación de base corresponde a Arduino UNO rev3 y por el otro lado, la estación remota corresponde al sistema diseñado basado en el ATtiny84.

Básicamente lo importante es este apartado son dos cosas. La primera es determinar o comprobar los niveles de tensión de corriente de alimentación del circuito. La segunda es comprobar el corrector funcionamiento de la unidad, básicamente que el chip funcione, y pueda trabajar con trenes de pulsos a 38kHz, esta velocidad será necesaria para acondicionar el sensor de infrarrojos.

3.1.1 Comprobando los valores de la batería

Punto	Voltaje	Intensidad máxima	Potencia
Bornes	3.70 V	750 mA	1.85 W

* Esta intensidad máxima se ha comprobado con los motores de continua, que son los elementos que mayor intensidad necesitan para funcionar.

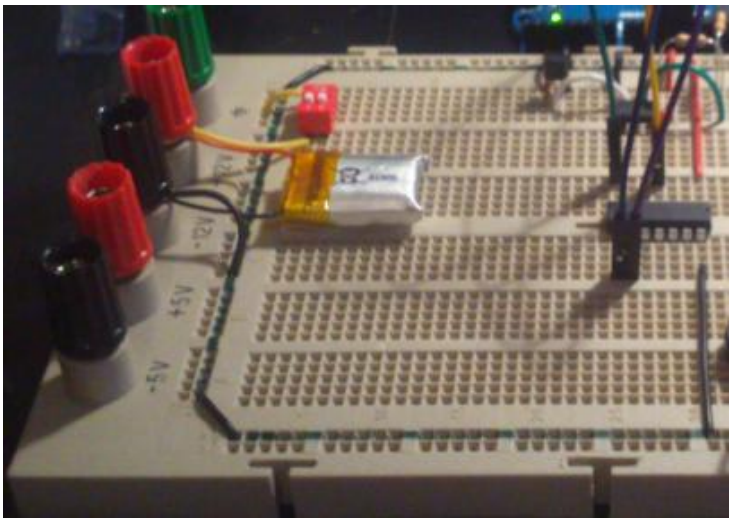


Fig. 3.1 Batería utilizada en el sistema.

3.1.2 Comprobando el conversor de carga

Punto	Voltaje	Intensidad máxima	Potencia
Entrada	3.70 V	750mA	2,775 W
Salida	5.05 V	200 mA	1,01 W

Primero vemos como la potencia de salida es menor, cosa lógica y esperada, porque ya sabíamos que tensión y que intensidad máxima nos proporcionaba el fabricante del MAX756.

Salida del conversor de carga con la configuración de elementos recomendados por el datasheet:

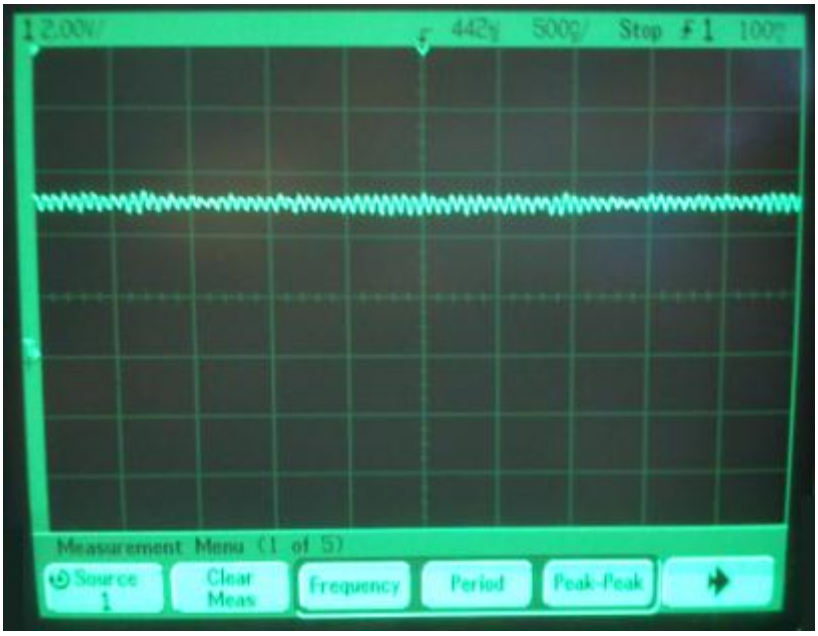


Fig. 3.2 Salida de 5V del MAX756 sin carga.

En este punto, parece que el sistema funciona. Ya que nos proporciona una señal estable a 5V. Ahora procedemos a conectar el ATtiny84 al conversor de carga. Lo que hacemos es cargar el programa Blink que se ha comentado en la figura 1.8 de la memoria.

El ATtiny carga correctamente el programa usando Arduino como ISP, y el LED parpadea. Procedemos entonces a conectar el ATtiny al conversor de carga. Una imagen de cómo se usa Arduino como ISP se puede ver en la figura 3.3.

El paso siguiente será conectar el ATtiny84 al conversor de carga. En primera instancia se observa que el sistema funciona correctamente.

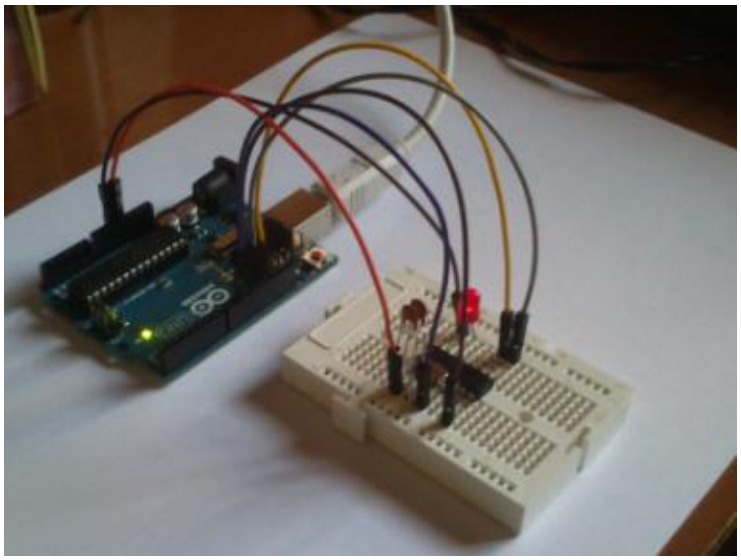


Fig. 3.3 Arduino como ISP cargando un programa en el ATtiny84.

A continuación se expone una foto del sistema y el programa cargado en el ATtiny84 para realizar las pruebas de funcionamiento.

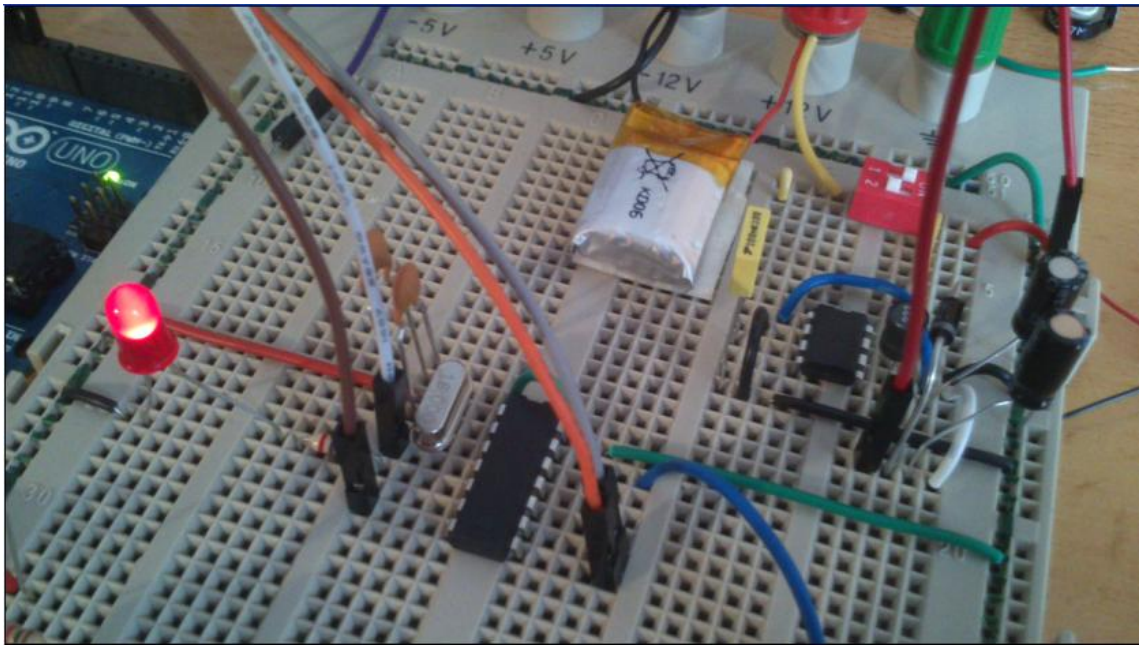


Fig. 3.4 MAX756 conectado al ATtiny84.

Programa cargado en el ATtiny para realizar pruebas de funcionamiento, en este caso se ha utilizado el pin digital 2, que corresponde al pin PB2.

Cómo ya hemos avanzado en la introducción de este capítulo, para adaptar el sensor de ultrasonidos, necesitaremos que nuestro microcontrolador sea capaz de generar una señal cuadrada a 38 kHz. Si no conseguimos este punto, nuestro sistema no podrá enviar información. Para comprobar que el sistema

funciona a una señal cuadrada de 38 kHz ± 1 kHz, cargaremos el siguiente programa en el ATtiny84:

```
int led = 2;      // Pin 2 -> PB2;
                  // NOTA: La configuración de pines puede variar según
                  // la librería utilizada.

/* Asignamos a una variable “led” el pin que queremos usar como salida
digital. PIN REAL -> PA0 */

void setup() {
  pinMode(led, OUTPUT);      // Asignamos el pin como salida
}
void loop() {
  digitalWrite(led, HIGH);   // Salida del Pin a VCC     -> LED ON
  delayMicroseconds(10);
  digitalWrite(led, LOW);    // Salida del Pin a 0V      -> LED OFF
  delayMicroseconds(10);
}
```

Fig. 3.5 Programa para realizar pulso de 38kHz

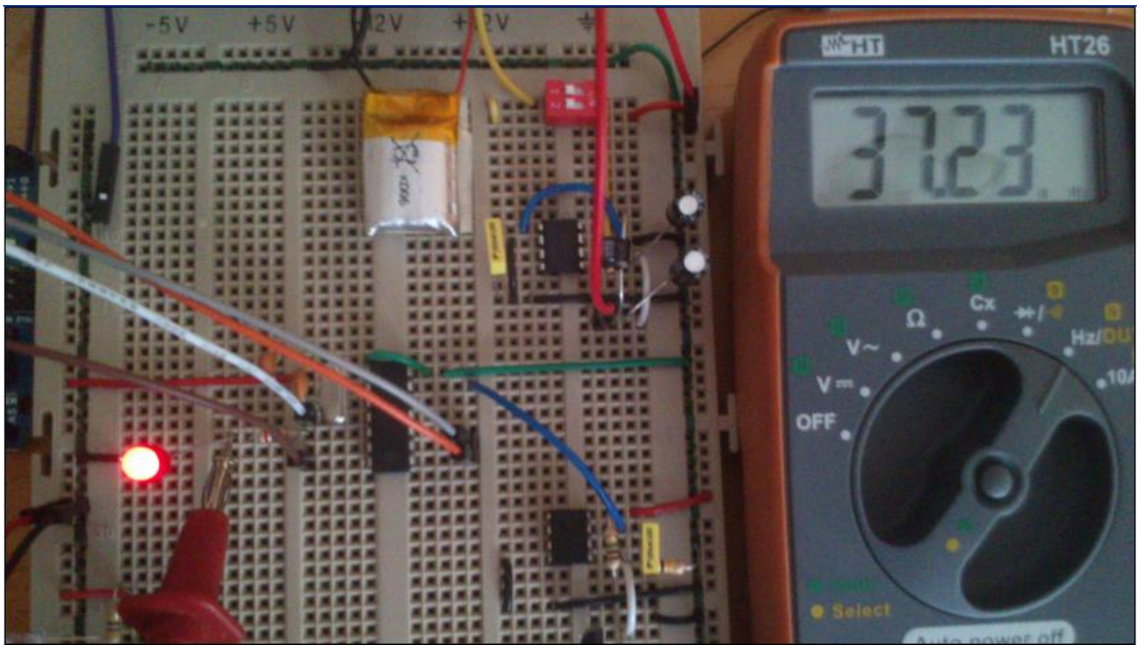


Fig. 3.6 Comprobación de onda cuadrada a 38kHz

Si nos damos cuenta, en verdad la generación de una cuadrada a 38 kHz, no corresponde a 10us ON y 10 us OFF, estos valores son valores experimentales, debido a que el microcontrolador trabaja a una velocidad establecida y no a la real. En realidad debido a que la frecuencia es la inversa del periodo nuestros pulsos deberían ser de 26,2 us.

Si realizamos la prueba con un osciloscopio veremos que conseguimos generar la siguiente onda cuadrada:

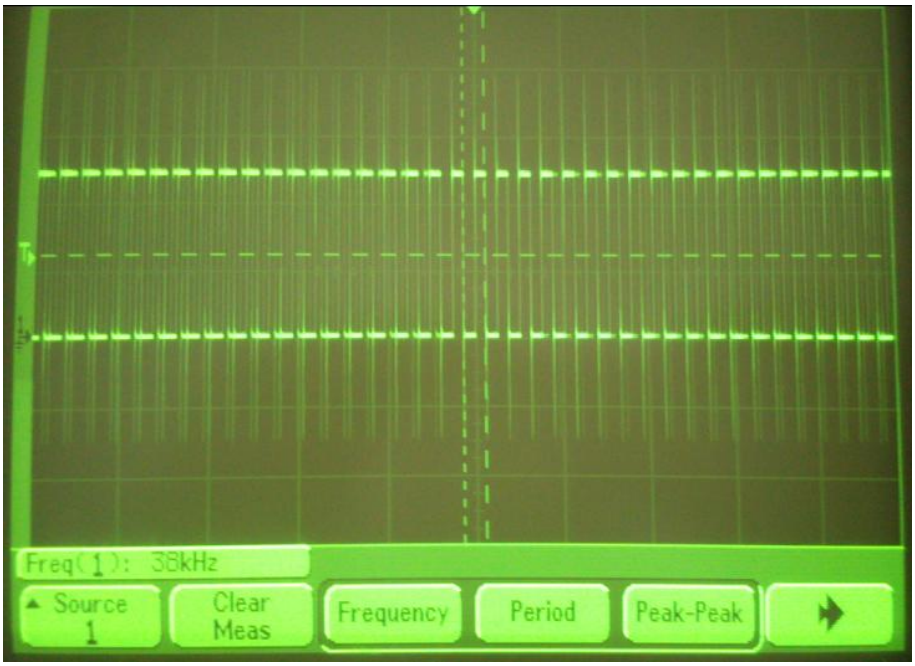


Fig. 3.7 Comprobación de onda cuadrada a 38kHz con osciloscopio.

Con esto concluye el aparatado de mediciones experimentales de nuestro ATtiny. Hemos sido capaces de generar un sistema remoto que puede trabajar a 5V con un coste de 11 euros. Aunque la batería no se considera en el precio, se puede encontrar baterías recargables para ser insertadas en PCB por menos de 5 euros. Un ejemplo, para futuros proyectos, se podría utilizar una batería de reducidas dimensiones de 2,4 V como la siguiente:

Tabla 3.1. Ejemplo batería insertable en PCB

Producto / REF/Link	Descripción	Precio Unitario
55615602940	Batería VARTA insertable en PCB, de 2,4V y 150mAh	4,36 €

En definitiva, por aproximadamente 15 euros, podemos tener un sistema totalmente autónomo, este precio es con elementos insertables en una PCB o una protoboard, por lo que vemos que para determinados usos no necesitaremos, por ejemplo, comprar una placa Arduino Mini, que aunque sus prestaciones son superiores (mayor número de pines, mayor memoria FLASH, etc.) solo la placa ya tiene un precio de 15 €*. Con esto no quiero dar una mala imagen a este tipo de placas, ni mucho menos, solo quiero dar a entender que dimensionando las necesidades de nuestro sistema podemos ahorrarnos coste de diseño.

* Precio Arduino Mini con cabeceras en store.arduino.cc 15/04/2013.

3.2 Sensor de efecto Hall

El objetivo planteado para este sensor, es ser capaces de detectar el Norte Magnético. En la figura 3.8 vemos cómo se comporta el sensor A1301 en presencia del campo magnético terrestre.

Fig. 3.8 Posición del sensor A1301 respecto a la intensidad de campo

Para realizar las pruebas del sensor de efecto Hall, se ha creado un pequeño programa en Arduino y en LabVIEW para monitorizar los resultados a la salida del sensor.

```
const int Hall_1 = A0;
int sensorCAL_1;

void setup() {
  Serial.begin(9600);
}

void loop() {

calibrar();

delay(100);
}

void calibrar(void)
{
  sensorCAL_1 = analogRead(Hall_1);

  Serial.println(sensorCAL_1);
}
```

Fig. 3.9 Programa de pruebas del sensor magnético

Cómo conclusión de las pruebas realizadas, se ha observado que es necesario enviar una media de 100 medidas para proporcionar valores de intensidad campo aceptables, si no es así la lectura es muy irregular ya que el sensor presenta una baja repetibilidad en su salida. El programa creado en LabVIEW ayuda mucho a entender cómo mediante métodos estadísticos (media, desviación estándar, etc) somos capaces de mejorar el acondicionamiento de un sensor.

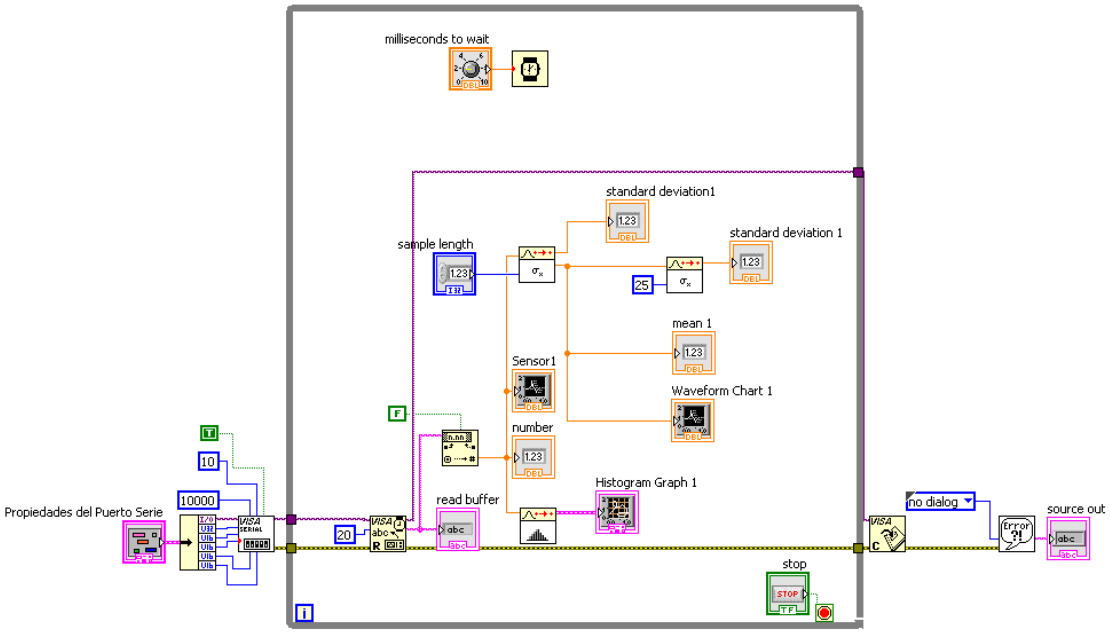


Fig. 3.10 Programa de pruebas en LabVIEW del sensor magnético

G	Desviación estándar máxima de la muestra	Valor ADC Hall apuntando al Norte	Valor ADC Hall apuntando al Sur
1000	3	965	743

La repetibilidad del sensor es baja además de que la respuesta a la dirección no es inmediata tarda alrededor de 2 segundos en estabilizarse la medida, pero se aprecia claramente la detección del norte magnético.

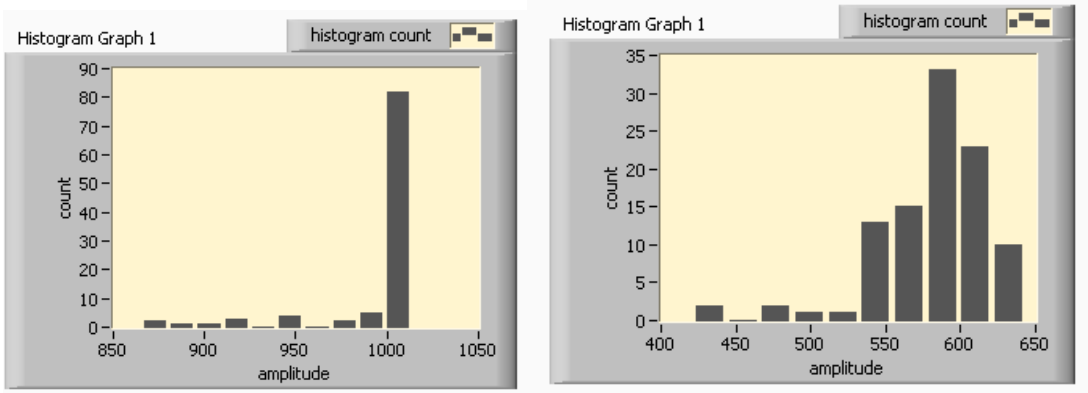


Fig. 3.11 Programa de pruebas en LabVIEW del sensor magnético

En el Anexo 6 “Esquemas de LabVIEW”, se puede encontrar cómo se ha realizado la calibración final del sensor A1301.

3.3 Sensor de infrarrojos

El acondicionamiento del sensor de infrarrojos, es una parte muy importante ya que es el transductor que nos permitirá la comunicación entre el Arduino y el ATtiny84.

Cómo esta parte del sistema se basa en dos elementos: emisor y receptor veremos ambos por separado.

3.3.1 Emisor de infrarrojos.

Como emisor de infrarrojos se ha escogido un OPE5685. La tabla de sus características principales se expone en la figura

ELECTRO-OPTICAL CHARACTERISTICS (Ta=25°C)

Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
Forward voltage	V_F	I_F=50mA		1.5	2.0	V
Reverse current	I_R	V_R=4V			10	μA
Capacitance	Ct	f=1MHz		20		pF
Radiant intensity	Ie	I_F=50mA		50		mW/sr
Peak emission wavelength	λp	I_F=50mA		850		nm
Spectral bandwidth 50%	Δλ	I_F=50mA		45		nm
Half angle	Δθ	I_F=50mA		±22		deg.
Optical rise & fall time(10%~90%)	tr/tf	I_F=50mA		25/13		ns
Cut off frequency *3	fc	I_F=50mA DC +10mA p-p		14		MHz

*3. 10logPo(fc MHz)/Po(0.1 MHz)=-3

Fig. 3.12 Principales características de un diodo emisor OPE5685. [7]

El OPE5685 es un diodo emisor que emite en una longitud de onda de 850 nm. La velocidad de cambio de estado de hasta 25 ns y un campo de misión de 22º. Cómo su nombre nos puede indicar su acondicionamiento será parecido al de un diodo común, con la diferencia que en este sistema, queremos hacer circular la máxima corriente posible (siempre hasta el máximo permitido), y en un LED común se hace todo lo contrario, evitar que la corriente llegue a determinados valores.

Si Arduino Uno nos proporciona como máximo 50mA es su pin 3v3. Aprovecharemos este para alimentar a nuestro OPE5685. En el caso del ATtiny84 como el MAX756 nos proporciona hasta 200mA, no tendremos problemas, pero para homogeneizar el sistema tanto el emisor del Arduino como el del ATtiny84 tendrán la misma configuración.

Cómo ya se ha comentado en la memoria principal el acondicionamiento del sensor será tal y como indica la figura 3.13.

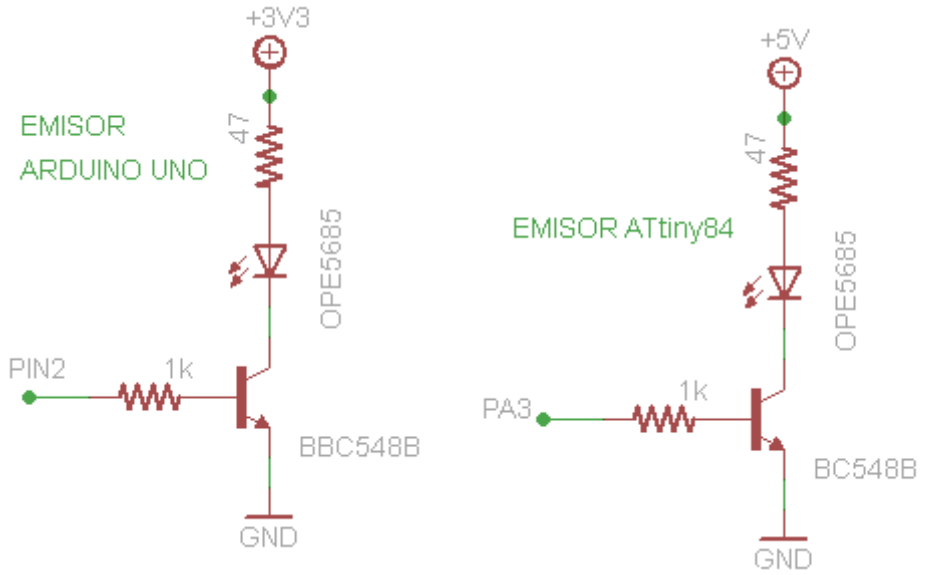


Fig. 3.13 Acondicionamiento emisor infrarrojos OPE5685.

La fórmula para acondicionar los valores de nuestro sensor es la siguiente:

$$R_c = {(VCC - 1.5V)}/{I_c} \quad \textbf{(3.1)}$$

Si tenemos o más bien queremos una intensidad de 50mA, tendremos una R_c de 70Ω en el ATtiny84 y de 36Ω en Arduino, como estos valores comercialmente son difíciles de encontrar, en ambos caso usaremos una resistencia de 47Ω. Para comprobar que el sistema funciona, usaremos un truco, muy útil cuando trabajemos con infrarrojos.

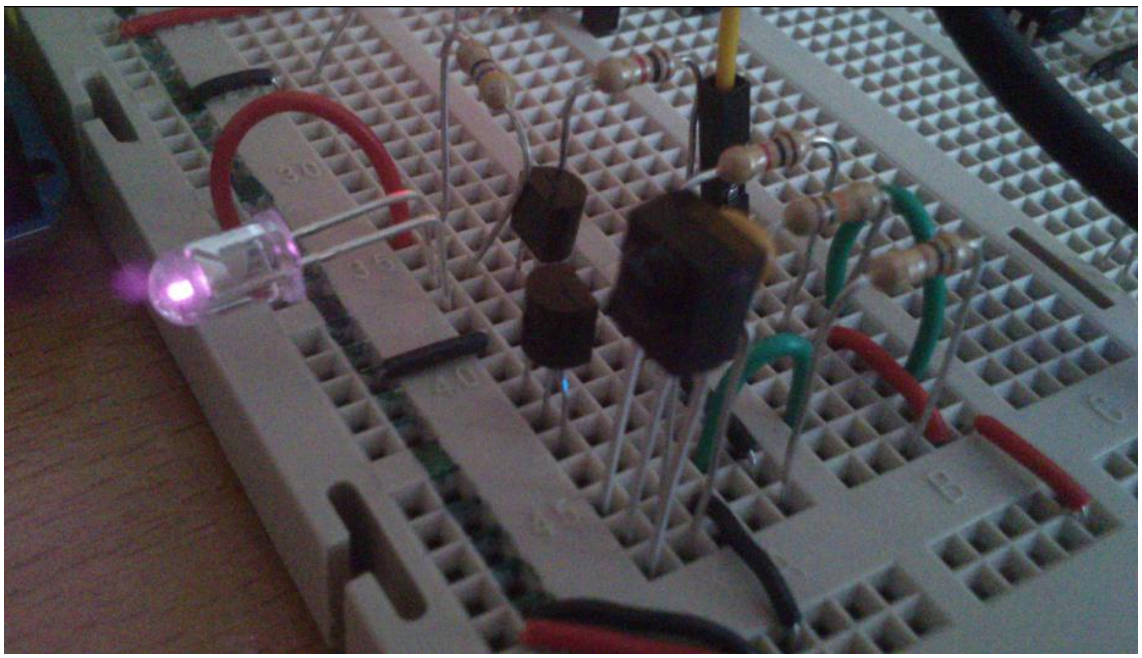


Fig. 3.14 Prueba de funcionamiento del OPE5685.

Cómo se ha comentado en la memoria, el ojo humano no es capaz de percibir la luz infrarroja, no obstante muchas cámaras sí que pueden, en concreto las cámaras de los móviles suelen funcionar muy bien para este objetivo.

Para comprobar el funcionamiento enviaremos la señal de 38kHz creada en los ejemplos anterior a través del sensor infrarrojos, está vez usaremos el puerto PA3 del Attiny84 y el pin digital 2 en el Arduino UNO. En la imagen 3.14 se puede ver el acondicionamiento del emisor de luz infrarroja.

3.3.2 Receptor de infrarrojos

El receptor de infrarrojos utilizado para el proyecto es el TSOP4P38. Las características principales de este receptor se muestran en la imagen siguiente:

ELECTRICAL AND OPTICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
Supply current (pin 3)	E_v = 0, V_S = 5 V	I_{SD}	0.65	0.85	1.05	mA
	E_v = 40 klx, sunlight	I_{SH}		0.95		mA
Supply voltage		V_S	2.7		5.5	V
Transmission distance	E_v = 0, test signal see fig. 1, IR diode TSAL6200, I_F = 400 mA	d		45		m
Output voltage low (pin 1)	I_{OSL} = 0.5 mA, E_e = 0.7 mW/m², test signal see fig. 1	V_{OSL}			100	mV
Minimum irradiance	Pulse width tolerance: $t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig. 1	$E_{e\ min.}$		0.17	0.35	mW/m²
Maximum irradiance	$t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig. 1	$E_{e\ max.}$	30			W/m²
Directivity	Angle of half transmission distance	$\varphi_{1/2}$		± 45		deg

Fig. 3.15 Principales características de un receptor TSOP4P38. [8]

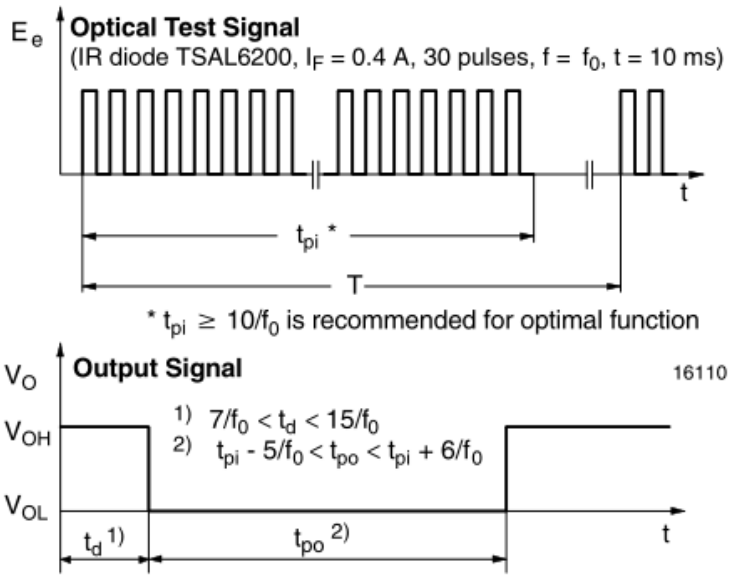


Fig. 3.16 Ejemplo de recepción en un TSOP4P38. [8]

Al igual que se ha hecho con el emisor, los circuitos de recepción serán idénticos tanto en el Arduino UNO como en el ATtiny84, exceptuando los pines de salida.

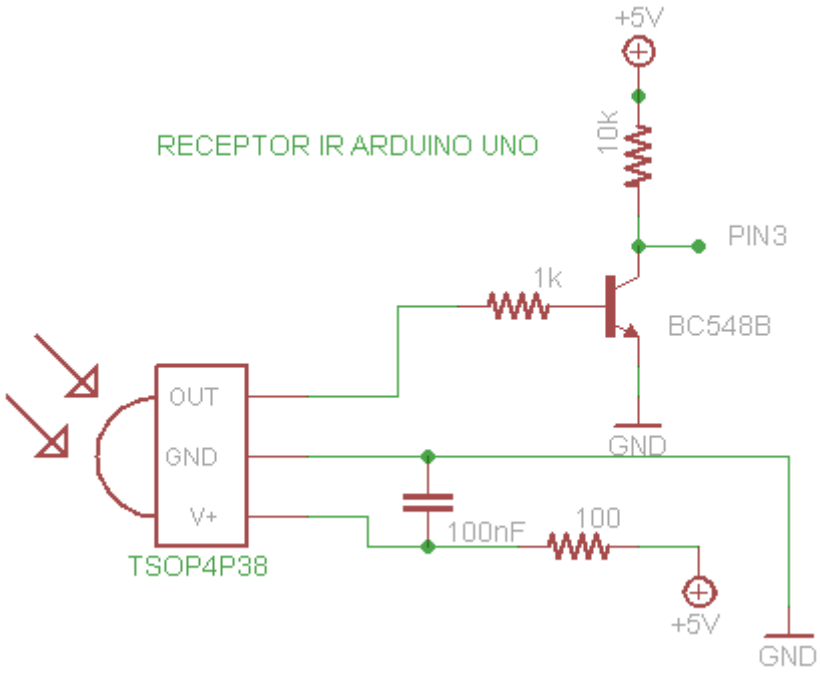


Fig. 3.17 Acondicionamiento del TSOP4P38 en el Arduino UNO.

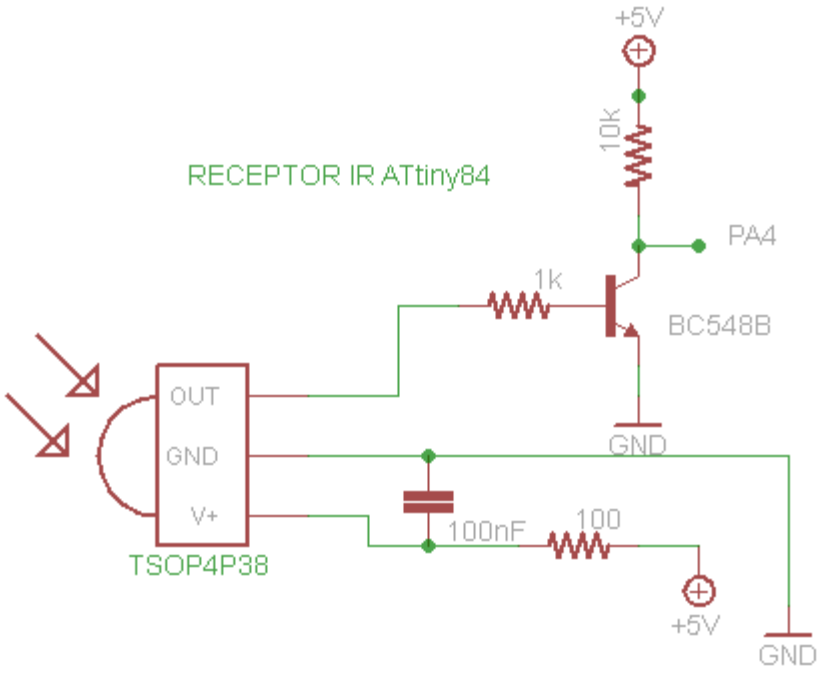


Fig. 3.18 Acondicionamiento del TSOP4P38 en el ATtiny84.

Para ver como se ha acondicionado este sistema, primero veremos cómo se comportaba el sensor según su datasheet. Lo primero que hacemos es enviar información a través del emisor y ver el comportamiento del TSOP4P38 en su salida. Para que el TSOP4P38 reciba algo debemos codificar un tren de pulsos a 38kHz. Para generar un tren de pulsos lo único que tenemos que hacer es una estructura *for* en nuestro programa de ejemplo.

```
int ledIR = 7;   // Pin 7 -> PA3;  OPE5685
                 // NOTA: La configuración de pines puede variar según
                 // la librería utilizada.

void setup() {
  pinMode(ledIR, OUTPUT);      // Asignamos el pin como salida
}
void loop() {

for(int i=0; i<24;i++)        // Tren de 24 pulsos a 38kHz.
{
  digitalWrite(ledIR, HIGH);
  delayMicroseconds(10);
  digitalWrite(ledIR, LOW);
  delayMicroseconds(10);
}

delayMicroseconds(500);        // Espera 500us
}
```

Fig. 3.19 Programa para enviar un tren de 24pulsos de 38kHz

La salida del TSOP será la siguiente:

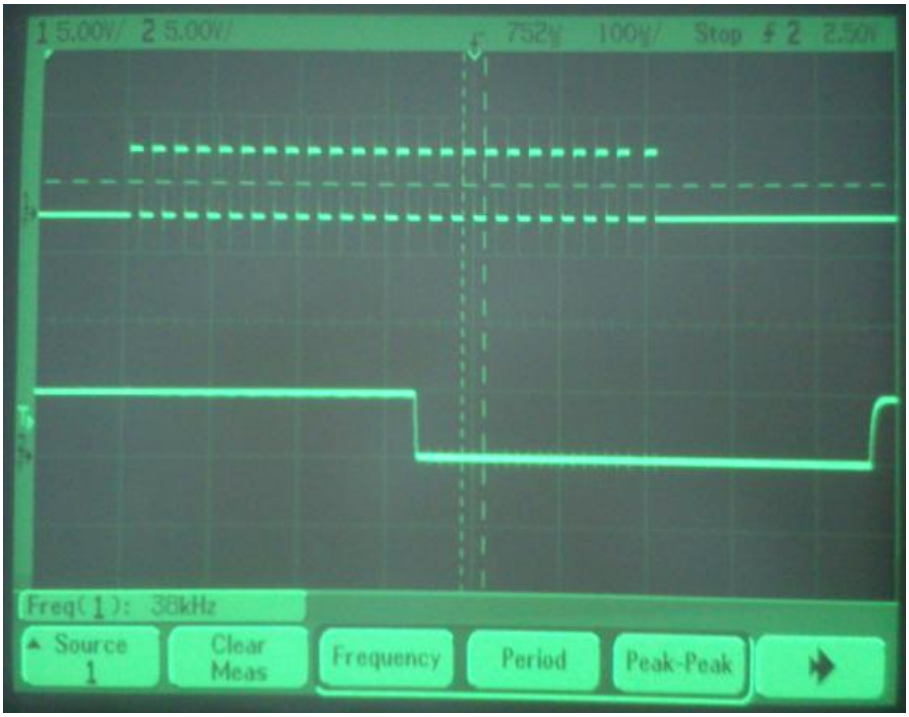


Fig. 3.20 Emisión y recepción de un tren de pulsos.

Cómo podemos observar, y tal cómo nos indica el fabricante, la salida del TSOP4P38 corresponde a un nivel bajo cuando detecta el tren de pulsos a 38kHz. Ahora bien, a la hora de detectar un pulso, nos podría interesar más que el comportamiento del sensor hiciese todo lo contrario, que cuando detectase un tren de pulsos pasase de 0 a 1. Esto nos facilitará nuestro trabajo a la hora de programar. Para realizar lo que se conoce como una puerta *not*, es decir poner VCC a la salida cuando en la entrada tengo 0V y viceversa.

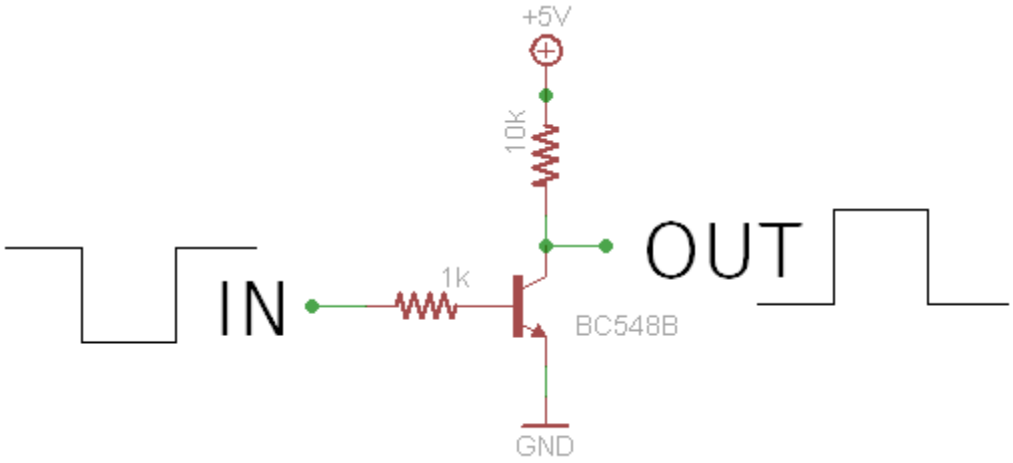


Fig. 3.21 Circuito de acondicionamiento del TSOP4P38

Salida después del acondicionamiento del TSOP4P38, en la imagen cada pico que aparece corresponde a un tren de pulsos.

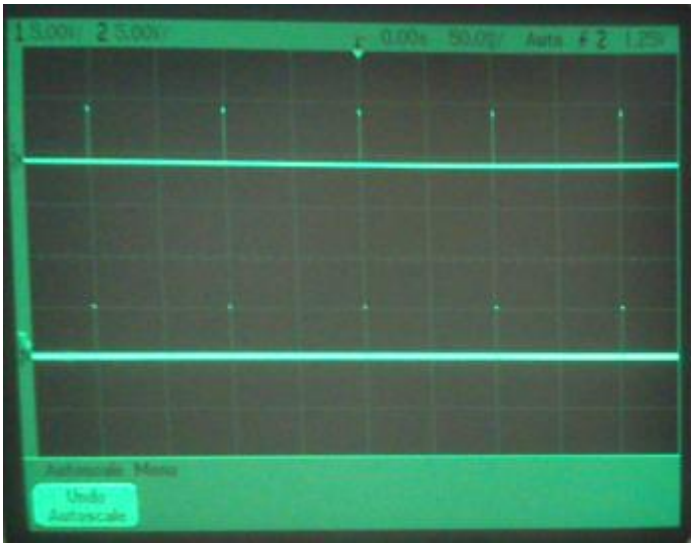


Fig. 3.22 Emisión y recepción de un tren de pulsos

ANEXO 4. COMPARATIVA SENSORES EFECTO HALL

Filtro Previo: Sensor Efecto Hall, Encapsulado DIP, Lineal, Bipolar

Allegro MicroSystems [@1] http://www.allegromicro.com/en/Products/Magnetic-Linear-And-Angular-Position-Sensor-ICs/Linear-Po

Modelo	Voltage Alimentación (V)	Voltage Salida (V)	Sensibilidad (mV/G)	Ancho de Banda (KHz)	Disponibilidad en Farnell	Referencia Farnell	Precio (sin IVA)
A1301	4,5 - 6	50% VCC	2,500	20	SI	A1301EUA-T	1,61 €
A1302	4,5 - 6	50% VCC	1,300	20	SI	A1302KUA-T	1,23 €
A1324	4,5 - 5,5	50% VCC	5,000	17	NO*		
A1325	4,5 - 5,5	50% VCC	3,125	17	SI**	A1325LUA-T	1,64 €
A1326	4,5 - 5,5	50% VCC	2,500	17	NO		
A1381	4,5 - 5,5	2,3-2,6 "P"	6-9 "P"	12	NO		
A1383	4,5 - 5,5	2,3-2,6 "P"	2,75-4.25 "P"	21	NO		
A1384	4,5 - 5,5	2,3-2,6 "P"	2-3 "P"	27	NO		
A1386	4,5 - 5,5	2,4-2,6 "P"	1,9-3,3 "P"	20	NO		

* Disponible en otro tipo de encapsulado

** Sin Stock

"P" Programable

Filtro Previo: Sensor Efecto Hall, Encapsulado DIP, Lineal, Sensibilidad entre 2,5 y 12 mV/G

Honeywell [@02] http://sensing.honeywell.com/

Modelo	Voltage Alimentación (V)	Voltage Salida (V)	Sensibilidad (mV/G)	Tiempo de Respuesta	Disponibilidad en Farnell	Referencia Farnell	Precio (sin IVA)
SS494B	4,5 - 10,5	2,50 ± 0,075	5,0 mV ± 0,4 mV/G	3 us	SI***	SS494B	2,67 €
SS495A	4,5 - 10,5	2,50 ± 0,075	3,125 mV ± 0,125 mV/G	3 us	SI	SS495A	2,43 €
SS495A1	4,5 - 10,5	2,50 ± 0,075	3,125 mV ± 0,094 mV/G	3 us	SI	SS495A1	3,49 €
SS496A	4,5 - 10,5	2,50 ± 0,075	2,500 mV ± 0,100 mV/G	3 us	SI***	SS496A	4,74 €
SS496A1	4,5 - 10,5	2,50 ± 0,075	2,500 mV ± 0,075 mV/G	3 us	SI***	SS496A1	3,22 €
SS496B	4,5 - 10,5	2,50 ± 0,075	2,500 mV ± 0,200 mV/G	3 us	NO		
SS49E	2,7 - 6,5	2,50 ± 0,075	1,4 mV/G	3 us	SI	SS49E	2,01 €
SS94B1	4,5 - 12	4,00 ± 0,04	3,125 mV ± 0,063 mV/G	3 us	NO		
SS94A1	6,6 -12,6	4,00 ± 0,04	5,0 mV ± 1 mV/G	3 us	SI	SS94A1	13,41 €
SS94A2	6,6 -12,6	4,00 ± 0,04	5,0 mV ±,1 mV/G	3 us	SI	SS94A2	20,62 €
91SS16-3	8,0 - 16,0	6,0 ± 0,6	9,9 mV ± 0,3 mV/G	3 ms	NO		

*** Disponible envio desde Newark (18€ de Portes de Envio)

ANEXO 5. PROGRAMACIÓN

En este capítulo se expondrán los dos programas cargados en los microcontroladores del proyecto. El programa cargado en el Arduino UNO se llama “Arduino_UNO.ino” y el programa cargado en el ATtiny84 “ATtiny.ino”

5.1 Arduino_UNO.ino

```
/*
Arduino_UNO.ino - Infrared comunication for ATtiny84
Copyright (c) 2013 Jorge Polo.  All right reserved.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

Licence: http://creativecommons.org/licenses/by-sa/3.0/
*/

/* Código de la librería ATtiny84:
http://code.google.com/p/arduino-tiny/

Bibliografía:
http://arduino.cc/es/Reference/Extended
http://www.cursomicros.com/avr/arquitectura/fuses-de-los-avr.html
http://www.engbedded.com/fusecalc/
http://codeandlife.com/2012/01/25/avr-attiny-usb-tutorial-part-2/
http://avrbasiccode.wikispaces.com/
http://www.fiz-ix.com/2012/11/arduino-serial-communication-bytes-
bases-and-ascii-characters/
*/

//*****************************************************************//
//*                          Arduino_UNO.ino                      *//
//*****************************************************************//

// Librerias:

// Definición de PINES:
const int RX = 2;          // Puerto Recepción Infrarrojos
const int TX = 3;          // Puerto Envío Infrarrojos

/*
   PROTOCOLO UTILIZADO: PROPIO
   CODIFICACIÓN DEL MENSAJE:

   CABECERA  MENSAJE  INFORMACION
   111       XXX      XXXXXXXXXX
   111       001      XXXXXXXXXX    ->   Sensor de Temperatura
*/

// Definición de Variables:

unsigned long duracion;
word cabecera = 0;
word tipoMensaje = 0;
word informacion = 0;
```

```
word mensajeRecibido = 0;
word mensaje_a_Enviar = 0;

double temp;

int byteSerial;

boolean error  = false;

void setup() {
  Serial.begin(115200);
  pinMode(TX, OUTPUT);
  pinMode(RX, INPUT);
}

void loop() {
enviarMensaje();
delay(50);
recibirMensaje();
delay(50);
}

void recibirMensaje(void)
{

mensajeRecibido = 0;
error = false;

for(int i=0; i<16; i++)
{
  duracion = pulseIn(RX, HIGH,150000);
                                  // Limita la velocidad de transmision
  if (duracion == 0)
  {
     break;
  }
  else if (duracion > 500)
                // Si el pulso dura más de 500 microsegundos, es un 1.
  {
      //Serial.print(1);
      bitSet(mensajeRecibido, 15-i);
  }
  else
  {
      //Serial.print(0);
  }
}

   Serial.println(mensajeRecibido,BIN);
                        // Imprime en binario el mensaje en el Serial.
}


void enviarMensaje(void)
{

mensaje_a_Enviar = 0;
error = false;

      for(int i=0; i<16; i++)
     {
```

```
          byteSerial = Serial.read();

          if (byteSerial == 49)
          {
            bitSet(mensaje_a_Enviar, 15-i);
          }
          else if (byteSerial == 48)
          {
            // No hacemos nada
          }
          else
          {
            error = true;
          }

// Si no leemos un cero o un uno es que ha habido un error en la
transmisión.

      }

    if (error)
    {
       //Serial.println("Error al enviar Mensaje");    //  Mensaje
codificado de error en transmisión
    }
    else
    {
       //Serial.println(mensaje_a_Enviar,BIN);  //  Imprime en binario
el mensaje en el Serial.
       transmitirMensaje(mensaje_a_Enviar);
    }
}


void enviarPulsoAlto() {
  // Función a realizar: 1. Envio un tren de 30 pulsos a 38 KHz.
  //                     2. Al finalizar espera 600 us.

    //  Serial.print("Enviando un 1 ");

      for (int i=0; i<30;i++)
      {
        digitalWrite(TX, HIGH);
        delayMicroseconds(9);
        digitalWrite(TX, LOW);
        delayMicroseconds(9);
      }

    delayMicroseconds(600);      // Espera de 600 microsegundos.
  }

  void enviarPulsoBajo() {
  // Función a realizar: 1. Envio un tren de 15 pulsos a 38 KHz.
  //                     2. Al finalizar espera 600 us.

  //  Serial.print("Enviando un 0 ");


    for (int i=0; i<15 ;i++)     // Tren de 15 pulsos a 38 KHz.
      {
        digitalWrite(TX, HIGH);
```

```
        delayMicroseconds(9);
        digitalWrite(TX, LOW);
        delayMicroseconds(9);
      }

    delayMicroseconds(600);                          // Espera de 600
microsegundos.
  }

void transmitirMensaje(word mensaje_a_Transmitir)
{

    int value = 0;

    for(int i=15;i>=0;i--)
    {
      value = bitRead(mensaje_a_Transmitir,i);
      if (value == 0)
        enviarPulsoBajo();
      else
        enviarPulsoAlto();
    }

}
```

5.2 ATiny.ino

```
/*
ATtiny.ino - Infrared comunication for ATtiny84
Copyright (c) 2013 Jorge Polo.  All right reserved.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

Licence: http://creativecommons.org/licenses/by-sa/3.0/
*/

/* Código de la librería ATtiny84:
http://code.google.com/p/arduino-tiny/

Bibliografía:
http://arduino.cc/es/Reference/Extended
http://www.cursomicros.com/avr/arquitectura/fuses-de-los-avr.html
http://www.engbedded.com/fusecalc/
http://codeandlife.com/2012/01/25/avr-attiny-usb-tutorial-part-2/
http://avrbasiccode.wikispaces.com/
*/

//*****************************************************************//
//*                          ATtiny.ino                           *//
//*****************************************************************//

// Librerias:
#include <SoftwareSerial.h>   // Libreria de puerto Serie - PROYECTOS
FUTUROS  -

// Definición de PINES:
const int RX      = 6;       // "PA4" Puerto Recepción Infrarrojos
const int TX      = 7;       // "PA3" Puerto Envío Infrarrojos
const  int  pinTemp  =  A0;          // "PA0" Puerto ADC del sensor de
temperatura
const int pinHall = A2;       // "PA2" Puerto ADC del sensor de efecto
HALL

//const int LED_ON = 9;          // "PA1" LED de Encendido del sistema
"Solo versiones de iniciales"

const int N = 100;             // Longitud Vector de datos del medidas
del sensor HALL
int sensorValue[N];           // Vector de Medidas del sensor de efecto
HALL

long sumaValue;                // Suma de Valores del sensor de efecto
HALL
float mediaValue;             // Media de Valores del sensor de efecto
HALL
```

```
int motor1 = 3;                 // "PB2" Control Motor de Cola Mov.
Derecha
int motor2 = 2;             // "PA7" Control Motor Principal 2
int motor3D = 4;            // "PA6" Control Motor Principal 1
int motor3I = 5;                // "PA5" Control Motor de Cola Mov.
Izquierda

/* Variables donde almacenaremos el estado anterior de los motores,
para evitar fluctuaciones. */

int motor1ANTERIOR  = 0;
int motor2ANTERIOR  = 0;
int motor3DANTERIOR = 0;
int motor3IANTERIOR = 0;

// Definición de Constantes:
word cabecera = 57344;                    //  EJEMPLO: 111 000 00 00 00
00 00

// Definición de Variables:
word tipoMensaje = 1024;                  //  EJEMPLO: 000 001 00 00 00
00 01
// Definición:
// 001 -> Información sobre temperatura
// 100 -> Motor principal 1
// 101 -> Motor principal 2
// 011 -> Motor de Cola

word informacion = 1023;                  //  EJEMPLO: 000 000 11 11 11
11 11

/*
   PROTOCOLO UTILIZADO: PROPIO
   CODIFICACIÓN DEL MENSAJE:

   CABECERA  TIPO MENSAJE  INFORMACION
   111       XXX           XXXXXXXXXX
   111       001           XXXXXXXXXX    -> Mensaje Temperatura
   111       010           XXXXXXXXXX    -> Mensaje Posicion
   111       100           XXXXXXXXXX    -> Motor Principal 1
   111       110           XXXXXXXXXX    -> Motor Principal 2
   111       011           XXXXXXXXXX    -> Rotor de cola Mov. Derecha
   111         101             XXXXXXXXXX     -> Rotor de cola Mov.
Izquierda
*/

word mensaje = 0;
int  infoTEMP = 0;              // Valor de temperatura
int  infoHALL = 0;              // Valor HALL

word mensajeRecibido = 0;
boolean error = false;
unsigned long duracion = 0;     // Duración del pulso recibido
boolean alternar = true;        // Intercalamos el envio de los dos
sensores.

int Z = 0;                      // Codifica Word a Int

void setup() {
  pinMode(TX, OUTPUT);
```

```
  pinMode(RX, INPUT);

  pinMode(motor1,   OUTPUT);       // Asignamos el pin como salida del
motor P.1
  pinMode(motor2,   OUTPUT);       // Asignamos el pin como salida del
motor P.2
  pinMode(motor3D, OUTPUT);        // Asignamos el pin como salida del
motor Cola
  pinMode(motor3I, OUTPUT);        // Asignamos el pin como salida del
motor Cola
}


void loop() {

//  Envio de información sobre sensores:
enviarInformacion();
delay(50);
//  Recibimos información sobre Motores:
recibirMensaje();
delay(50);


delay(150);
}

//*************************************************************//

void enviarInformacion()
{

if (alternar)
{
  // Envio de información sobre temperatura:

  infoTEMP = analogRead(pinTemp);
  cabecera    = 57344;  // "111" Cabecera
  tipoMensaje = 1024;   // "001" Temperatura
  informacion = word(infoTEMP);
  mensaje = cabecera | tipoMensaje | informacion;

  transmitirMensaje(mensaje);
  alternar = false;
}
else
{
  // Envio de información sobre sensor efecto HALL:

    //infoHALL  =  analogRead(pinHall);    //  Solo  para  calibrar  el
potenciomentro del sensor.

    // Cuando ya el potenciometro ya está calibrado, se envia la media
de los valores:

    for (int i=0; i<N;i++)    // Promediamos la salida -> Atenuamos el
ruido.
    {
        sensorValue[i] = analogRead(pinHall);
    }

    sumaValue = 0;
```

```
    for (int i=0; i<N;i++)
    {
         sumaValue = sensorValue[i] + sumaValue;
    }

    mediaValue = 0;
    mediaValue = int (float(sumaValue) / N);

    cabecera    = 57344;  // "111" Cabecera
    tipoMensaje = 2048;   // "010" HALL
    informacion = word(mediaValue);
    mensaje = cabecera | tipoMensaje | informacion;

    transmitirMensaje(mensaje);

    alternar = true;
}


}


void enviarPulsoAlto() {
// Función a realizar: 1. Envio un tren de 30 pulsos a 38 KHz.
//                     2. Al finalizar espera 600 us.

   for (int i=0; i<30;i++)
   {
      digitalWrite(TX, HIGH);
      delayMicroseconds(10);
      digitalWrite(TX, LOW);
      delayMicroseconds(10);
    }

    delayMicroseconds(600);
}

void enviarPulsoBajo() {
// Función a realizar: 1. Envio un tren de 15 pulsos a 38 KHz.
//                     2. Al finalizar espera 400 us.

   for (int i=0; i<15 ;i++)       // Tren de 30 pulsos a 38 KHz.
   {
      digitalWrite(TX, HIGH);
      delayMicroseconds(10);
      digitalWrite(TX, LOW);
      delayMicroseconds(10);
   }

   delayMicroseconds(600);
}

void transmitirMensaje(word mensaje_a_Transmitir)
{
  // Función a realizar: 1. Envia un mensaje de 16 bits -> "Envia 0 o
1 según corresponda"
  //                     2. Al finalizar espera 1000 us.
    int value = 0;

    for(int i=15;i>=0;i--)
```

```
    {
      value = bitRead(mensaje_a_Transmitir,i);
      if (value == 0)
        enviarPulsoBajo();
      else
        enviarPulsoAlto();
    }
}

void recibirMensaje(void)
{

mensajeRecibido = 0;

for(int i=0; i<16; i++)
{
  duracion = pulseIn(RX, HIGH,150000);
  if (duracion == 0)
  {
     break;
  }
  else if (duracion > 500)            // Si el pulso dura más de 500
microsegundos, es un 1.
  {
      //Serial.print(1);
      bitSet(mensajeRecibido, 15-i);
  }
  else
  {
      //Serial.print(0);
  }

  codificarMensaje(mensajeRecibido);
}

}

void codificarMensaje(word mensaje)
{
    cabecera = 0;
    tipoMensaje = 0;
    informacion = 0;

    int value = 0;
    int j = 0;
    int elegir = 0;
    int cabeceraOK = 0;

    for(int i=13;i<=15;i++)
    {
      value = bitRead(mensaje,i);
      if (value == 1)
          bitSet(cabecera, j);
      j++;
    }

    j = 0;
    for(int i=10;i<=12;i++)
    {
      value = bitRead(mensaje,i);
      if (value == 1)
```

```
          bitSet(tipoMensaje, j);
      j++;
    }

    j = 0;
    for(int i=0;i<=9;i++)
    {
      value = bitRead(mensaje,i);
      if (value == 1)
          bitSet(informacion, j);
      j++;
    }

    Z      = int(informacion);
    elegir = int (tipoMensaje);
    cabeceraOK = int (cabecera);

    if (cabeceraOK == 7)
    {
      switch (elegir)
      {
        case 4:  // 100
             if (Z != motor1ANTERIOR)
             {
               analogWrite(motor1, Z);
               motor1ANTERIOR = Z;
             }
        break;

        case 6:  // 110
             if (Z != motor2ANTERIOR)
             {
               analogWrite(motor2, Z);
               motor2ANTERIOR = Z;
             }
        break;

        case 3:  // 011
             if (Z != motor3DANTERIOR)
             {
               analogWrite(motor3D, Z);
               analogWrite(motor3I, LOW);
               motor3DANTERIOR = Z;
             }

        break;

        case 5:  // 101
             if (Z != motor3IANTERIOR)
             {
               analogWrite(motor3I, Z);
               analogWrite(motor3D, LOW);
               motor3IANTERIOR = Z;
             }

        break;
        default:
        break;
      }
    }
}
```

ANEXO 6. ESQUEMAS DE LABVIEW

El diseño de la interfaz gráfica se presenta en la figura 6.1.

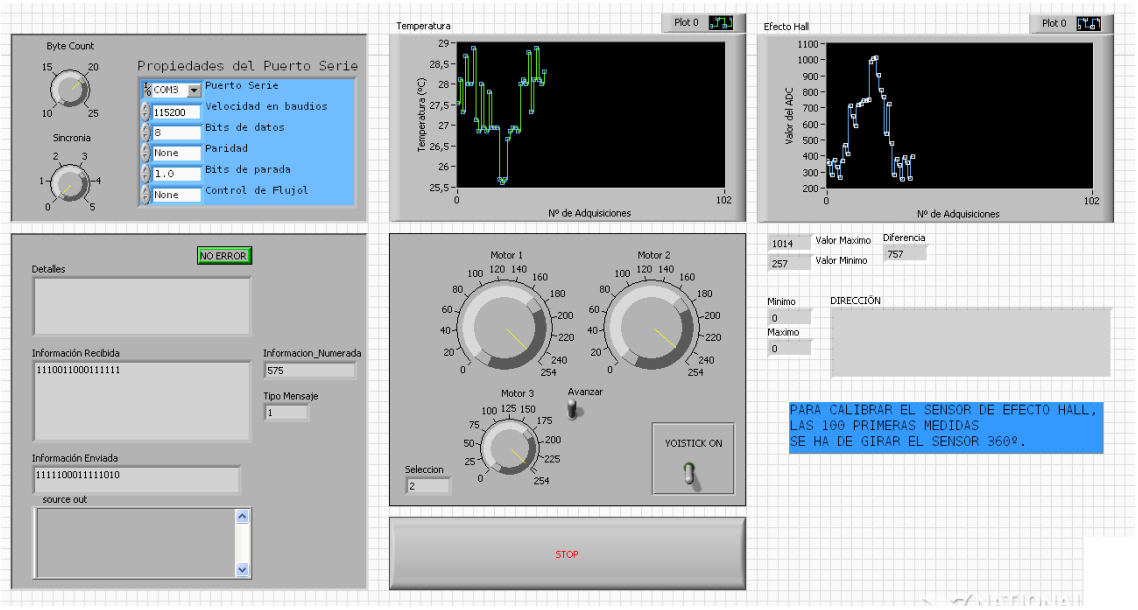


Fig. 6.1 Pantalla de control de la interfaz gráfica

El esquema completo del programa se presenta en la figura 6.2.

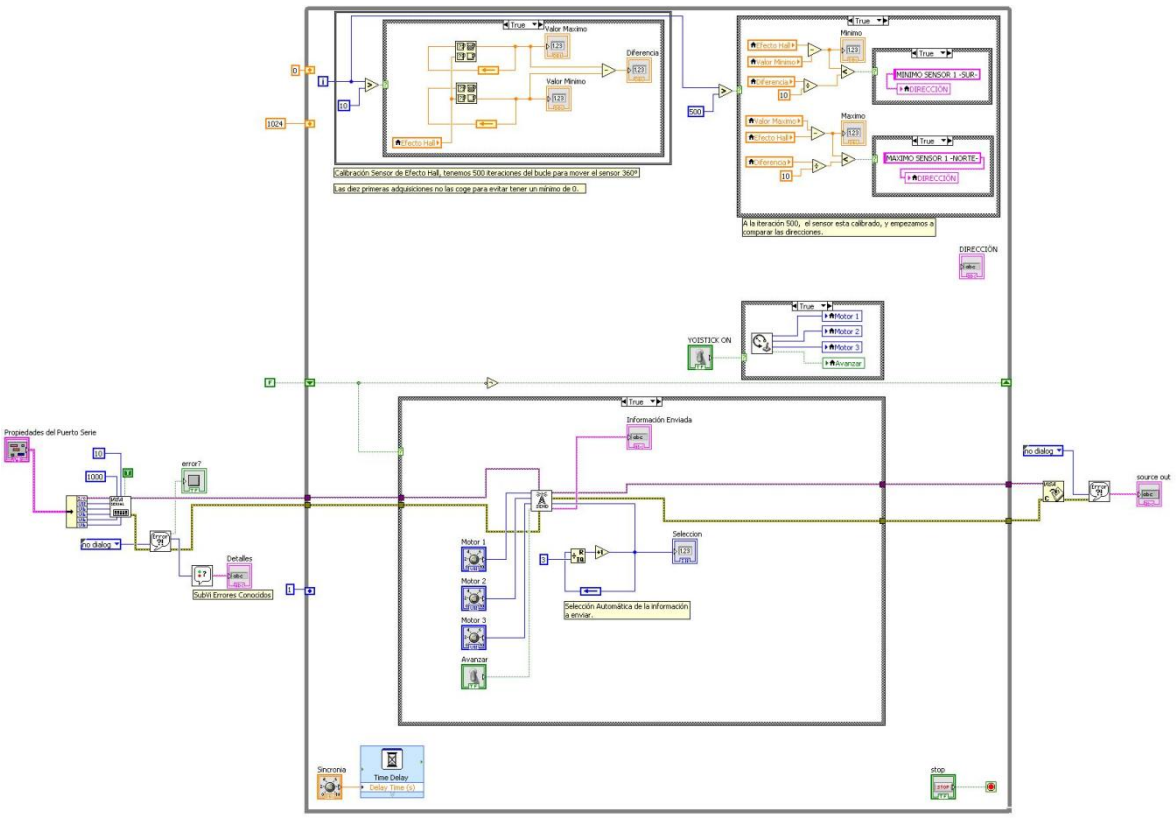


Fig. 6.2 Programa completo del programa realizado en LabVIEW

Una vez visto los esquemas generales del sistema, vamos a ver uno por uno cada uno de los módulos que lo forman. En primer lugar vamos a ver el control de puerto serie. En la figura 6.3 tenemos ampliada la zona de las propiedades y un pequeño SubVI o subrutina que se ha realizado para el control de los errores conocidos.

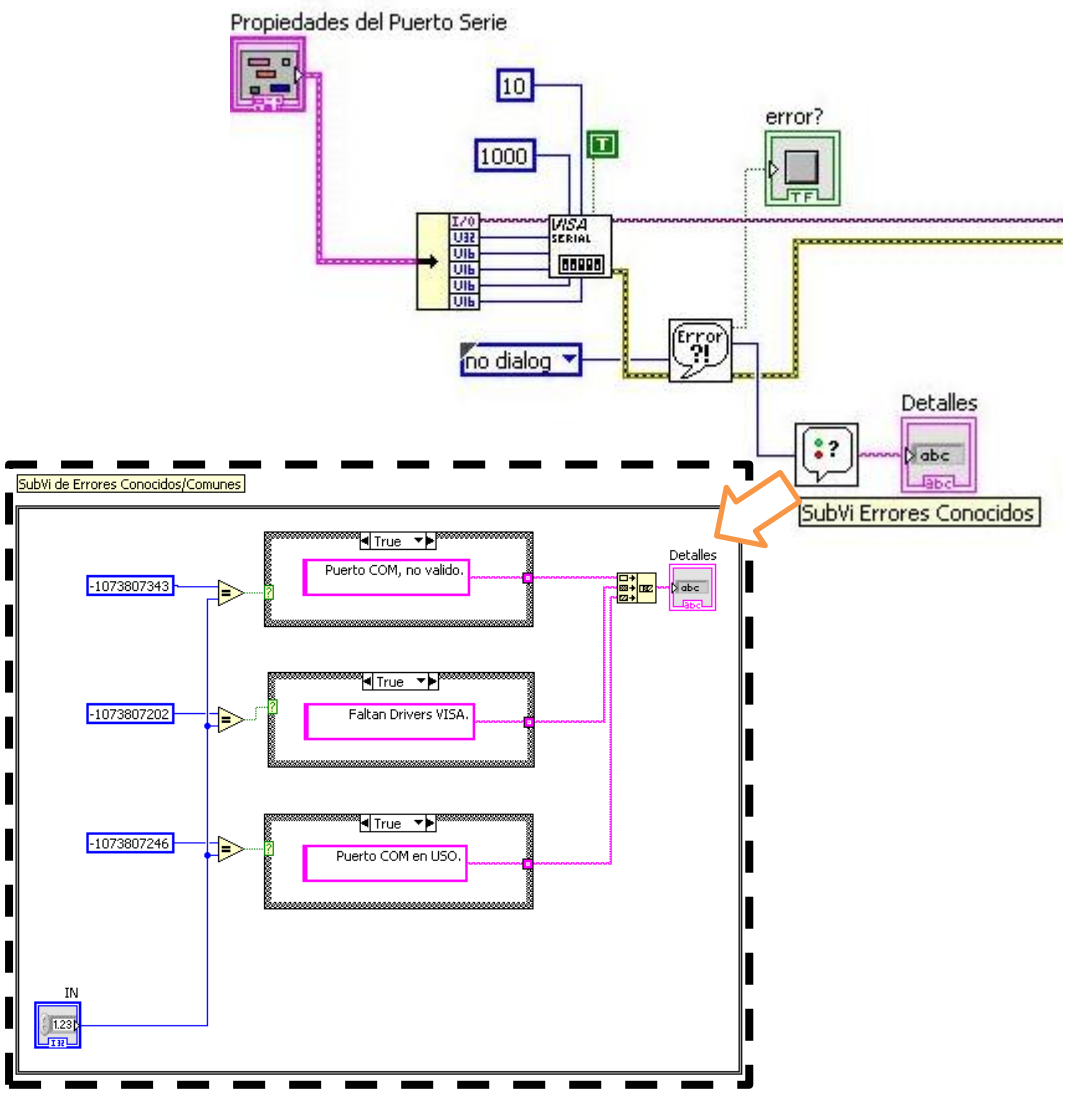


Fig. 6.3 Propiedades del puerto Serie y Control de Errores Conocidos.

La siguiente parte del programa es un bucle *while,* que contiene el núcleo del programa. Al igual que en la parte de programación de los microcontroladores, esta parte, recibe y envía información alternativamente en cada secuencia del bucle. El bucle finaliza cuando el usuario presiona el botón de parada en la pantalla de control.

Básicamente el bucle se va repitiendo, y en cada iteración una variable booleana (variable que solo puede ser verdadera o falsa) actúa como mecanismo de control para intercambiar entre el envío y la recepción de la información, si es verdadera se envía y si es falsa se recibe información. Esta

variable se niega en cada iteración para que en cada iteración el valor contenga un valor opuesto al anterior.

Cuando la variable es verdadera enviamos información. La información que tenemos que enviar es la velocidad de los motores. Para ello se ha creado dos pequeños subprogramas o SubVIs, que recogen la posición de los controles de los motores y codifican la información a enviar. Cada vez que se entra a enviar información, se envía información de un motor diferente.

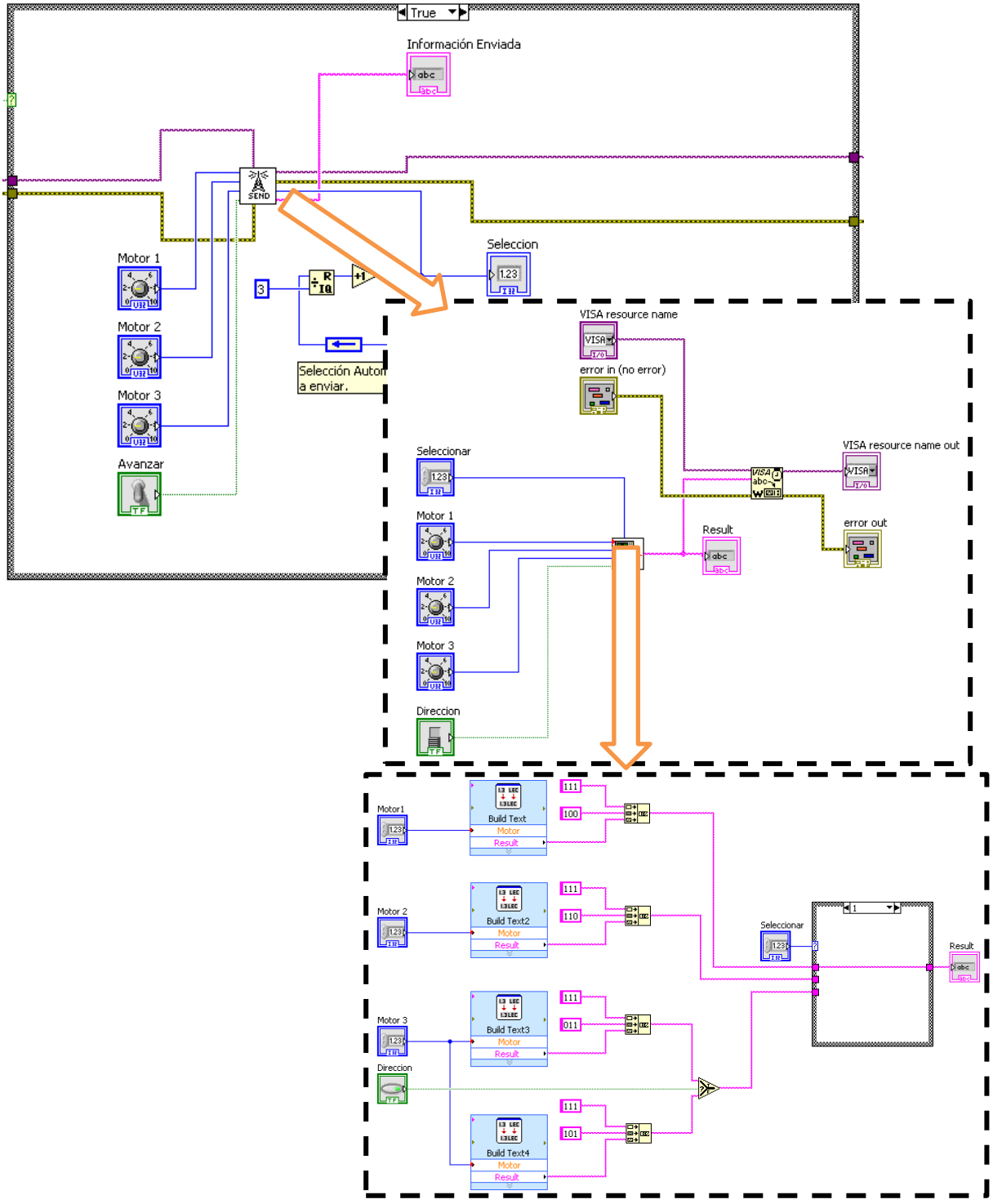


Fig. 6.4 Enviado información desde LabVIEW al puerto serie

En la figura 6.5 se puede ver el procedimiento a seguir para enviar información. Básicamente lo que se hace es coger la información de los controles de cada motor desde la pantalla de control y codificar la información, mediante el protocolo que se ha creado anteriormente, por lo que a cada motor se le añade una cabecera y un cadena del tipo de mensaje. Una vez se ha creado la cadena se envía al puerto serie para que Arduino UNO pueda enviarla. Para cambiar entre los diferentes motores, se ha hecho de forma automática, cada vez que el programa entra en el procedimiento enviar, una variable varía su valor desde 1 hasta 3, este número indica a que motor le toca enviar la información.

La otra parte del bucle, es cuando recibimos información.

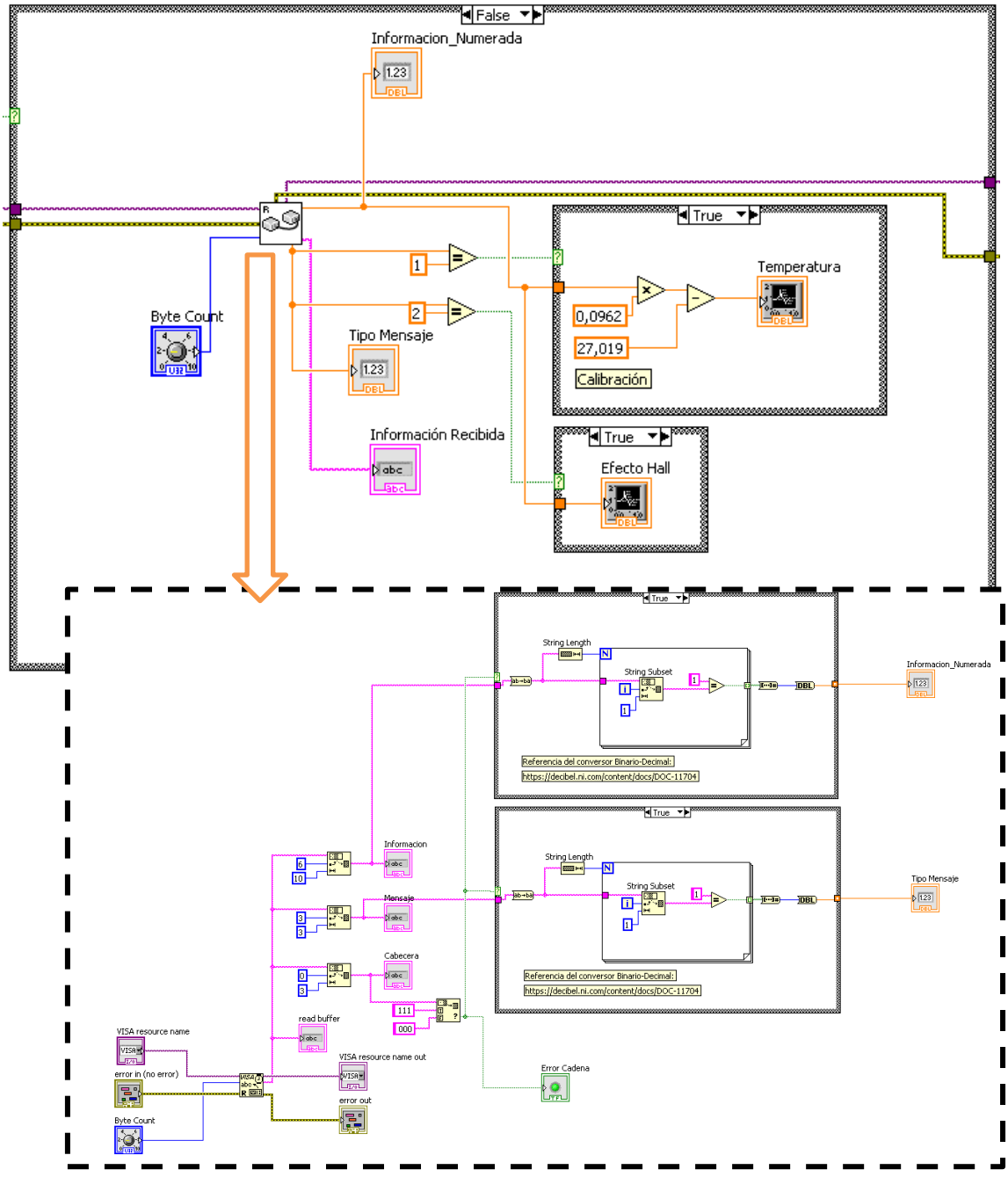


Fig. 6.5 Enviado información desde LabVIEW al puerto serie

En esta parte del programa lo que hacemos, es recoger una cadena de caracteres desde el puerto serie. Cómo hemos establecido un carácter de fin de línea en las propiedades del puerto serie, normalmente estas cadenas vendrán separadas y serán cadenas de unos y ceros de 16 posiciones.

Lo que hacemos es trocear la cadena, en elementos útiles, por ejemplo los tres primeros bits de la cadena son los bits de cabecera, los tres bits siguientes son los bits que nos indican que tipo de mensaje estamos recibiendo y por último, los diez últimos bits es donde viene codificada la información. Una vez troceada la cadena sólo tenemos que transformar la cadena de formato binario a decimal. Con la información en decimal ya podemos clasificar los mensajes y extraer la información. El esquema de como recibimos información mediante LabVIEW se representa en la figura 6.5.

Una vez recibida la información, tenemos dos tipos de mensajes: información de temperatura e información del sensor de efecto Hall. El primero de los mensajes que nos podemos encontrar, es la información sobre temperatura, esta información se codifica con una cadena de “001” o lo que es lo mismo, un uno en formato decimal. La segunda tipo de información que nos podemos encontrar es la información del sensor de efecto Hall, esta se codifica con un “010” o un dos en formato digital. Como se puede apreciar en la figura 6.6 lo único que hacemos es comparar el tipo de mensaje entrante con un dos o con un uno.

La siguiente parte del programa es la calibración de los sensores. En el caso del sensor de temperatura es muy sencillo, ya que solo tenemos que aplicar los factores de la curva de calibración.

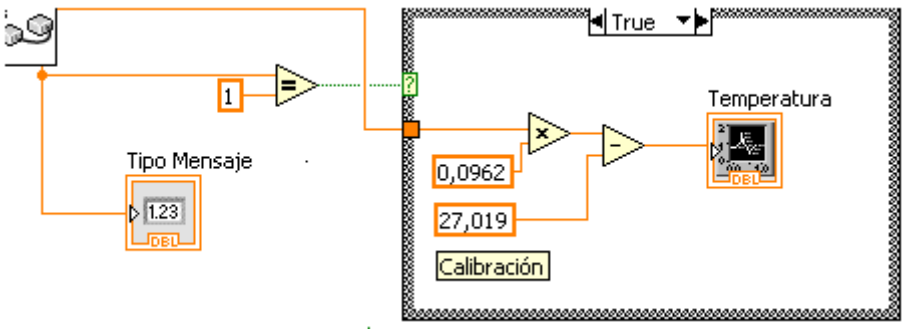


Fig. 6.6 Calibración del sensor de temperatura LM35DZ

En cuanto a calibrar el sensor de efecto Hall es un poco más complicado. Lo que debemos hacer es una serie de mediciones iniciales, moviendo el sensor 360º para captar la máxima y mínima intensidad de campo. El sensor tiene un único sentido donde la intensidad de campo es máxima, este punto corresponderá con el Norte magnético.

Fig. 6.7 Posición del sensor A1301 respecto a la intensidad de campo

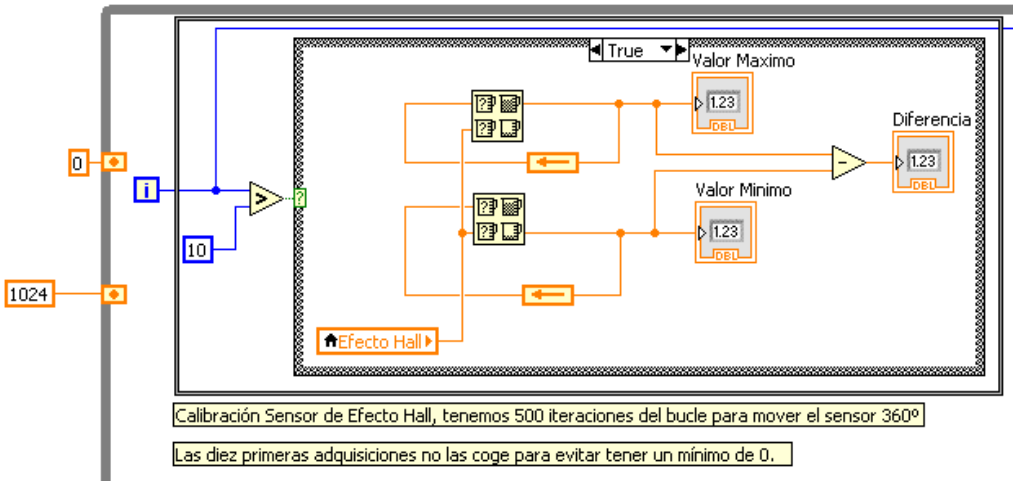


Fig. 6.8 Método de calibración del sensor A1301

Para calibrar el sensor lo que hacemos es buscar el valor máximo y mínimo en las primeras 500 iteraciones del bucle, que corresponderán a unas 115 mediciones de campo. Hay que tener en cuenta que para atenuar el ruido, el ATtiny84 no nos envía el valor de una única medición sino que nos envía la media de 100 mediciones. En la figura 6.8 se puede ver el código para encontrar el valor máximo y el mínimo de una medida en un bucle. En la figura se puede apreciar que no se empieza a buscar el máximo y el mínimo hasta la iteración 10 del bucle, esto se ha hecho para evitar el valor 0 o las fluctuaciones de las primeras mediciones.

Una vez se llega a las 500 iteraciones o las 115 mediciones de campo, para saber si estamos en dirección al norte magnético, lo único que tenemos que hacer es comparar nuestra medición con el máximo de campo registrado. En la figura 6.10 se puede ver el código utilizado para calcular la posición. Debido a la poca repetibilidad del sensor, se ha puesto un margen de error, que corresponde a la diferencia entre el valor máximo y el mínimo partido entre 10.

Cómo último parte del programa, se ha añadido un módulo que permite el control de los motores a través de un joystick. Ese módulo se ha basado en un ejemplo que incluye LabVIEW para el control de periféricos y se ha adaptado a un joystick “Logitech Force 3D Pro”. Gracias a esta ampliación el control de los motores es mucho más cómodo y preciso. Para poder usar este tipo de periféricos al sistema, se ha calibrado el joystick y se ha convertido su señal de salida en valores entre 0 y 255, valores adaptados para el control de los motores del proyecto.

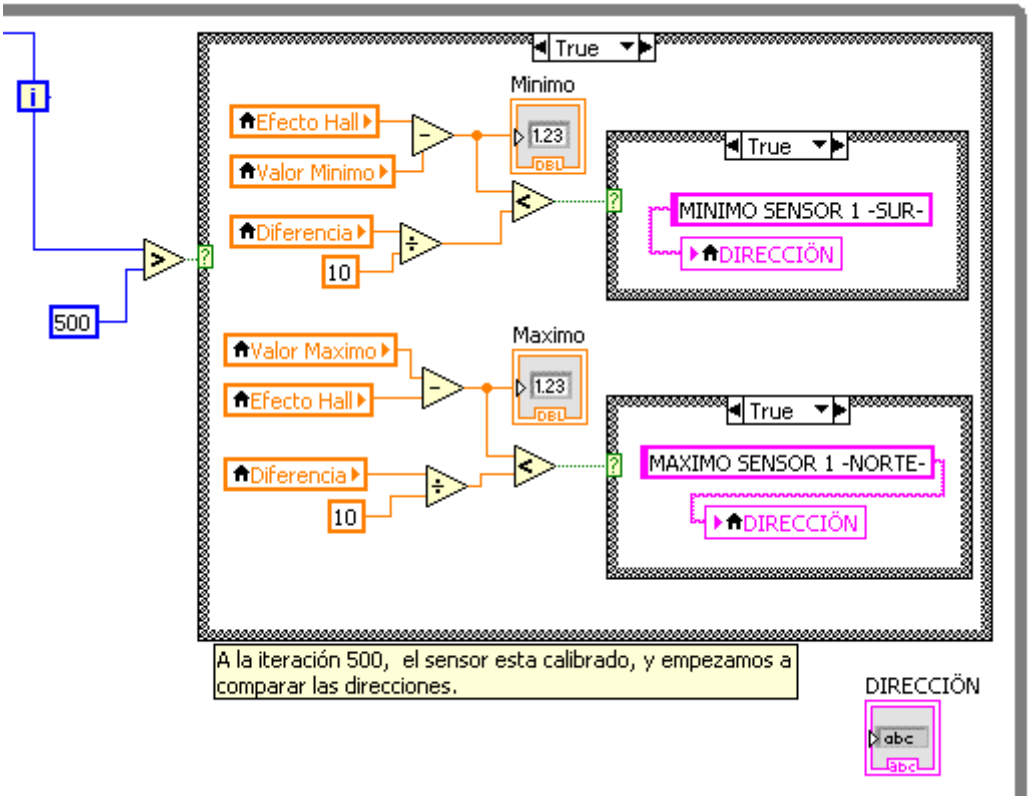


Fig. 6.9 Método de detección del Norte Magnético

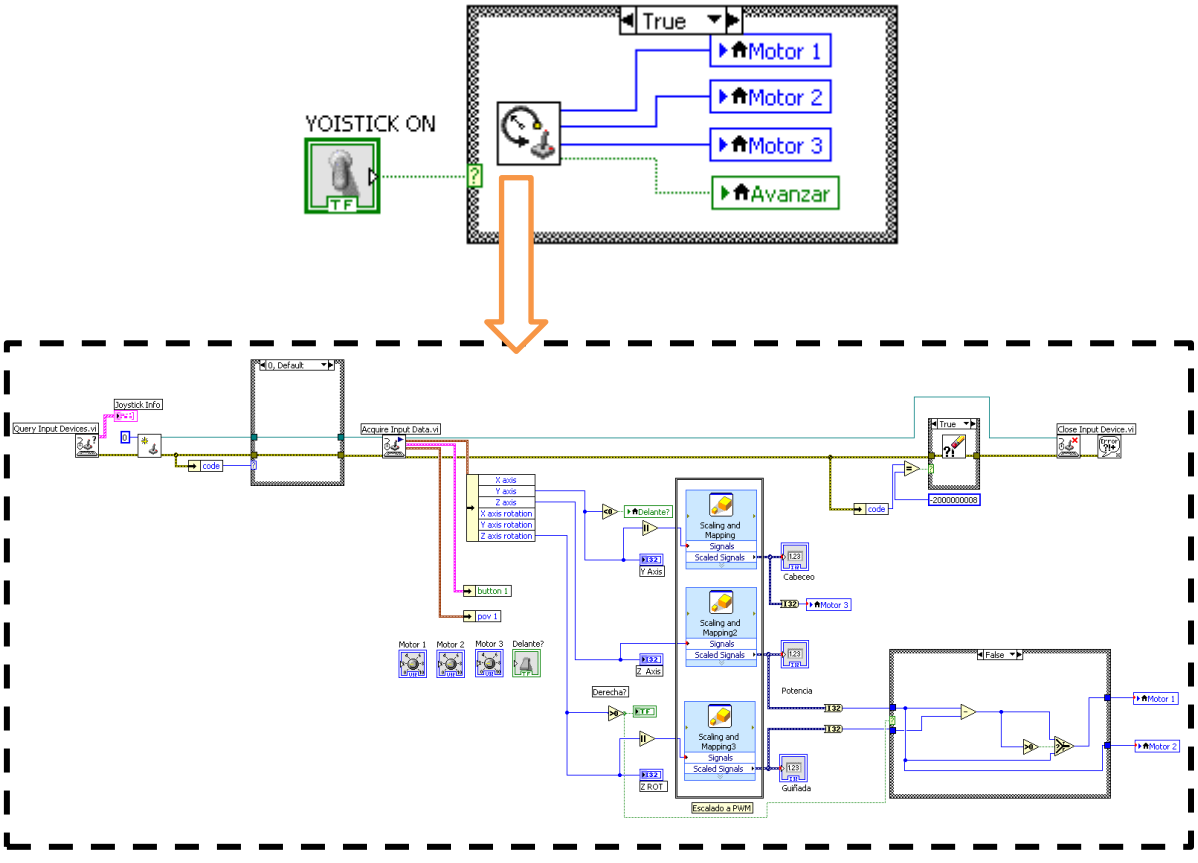


Fig. 6.10 Código para añadir un joystick al sistema

Cómo última parte, lo único que queda por hacer es una vez acabado el bucle por demanda del usuario, deberemos cerrar el puerto serie, para permitir a otros programas poder utilizarlo. El código para cerrar el puerto serie se representa en la figura 6.11.

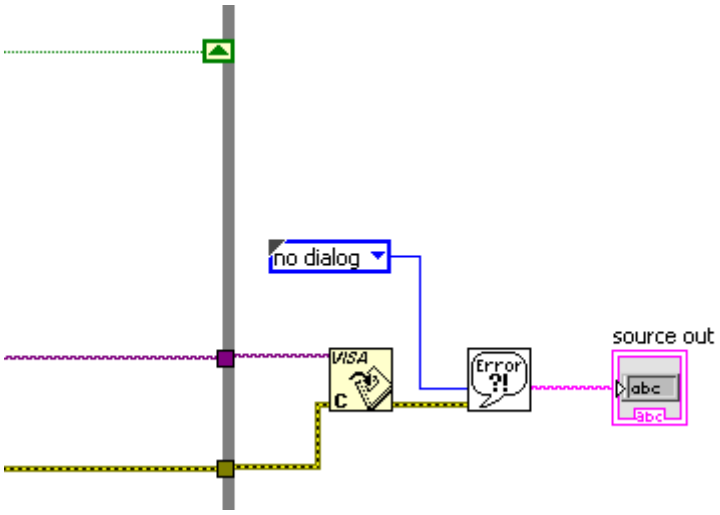


Fig. 6.11 Código para cerrar el puerto serie

ANEXO 7. SISTEMA

Cómo conclusión del sistema, se presenta el esquemático completo de la unidad remota y el diseño de una placa de circuito integrado de dimensiones reducidas (50 x 49 mm).

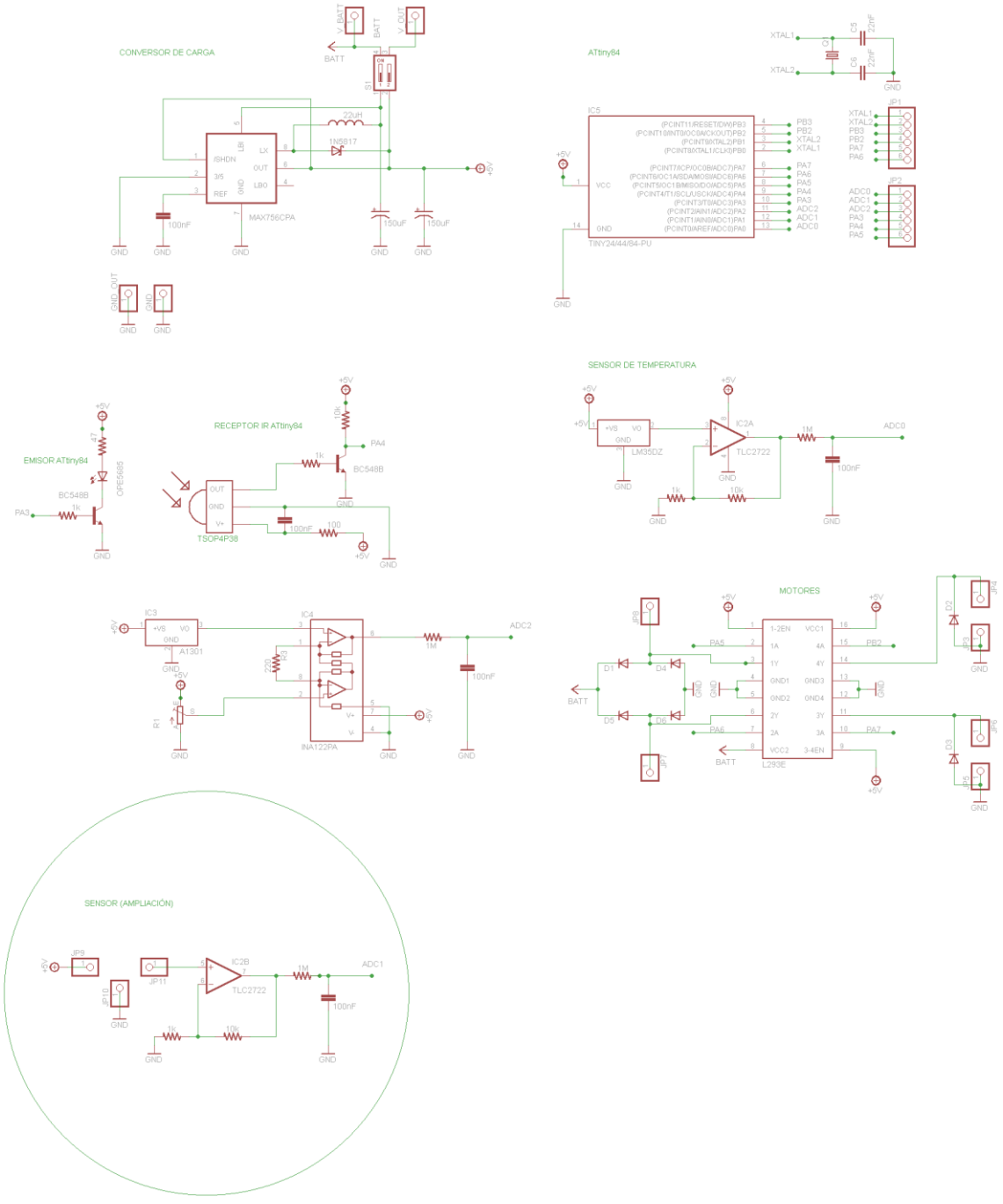


Fig. 7.1 Sistema completo de estación remota

Cómo el amplificador operacional TLC2272, solo tenía implementado el LM35DZ, se ha diseñado un pequeño circuito para la entrada sobrante.

7.1 PCB

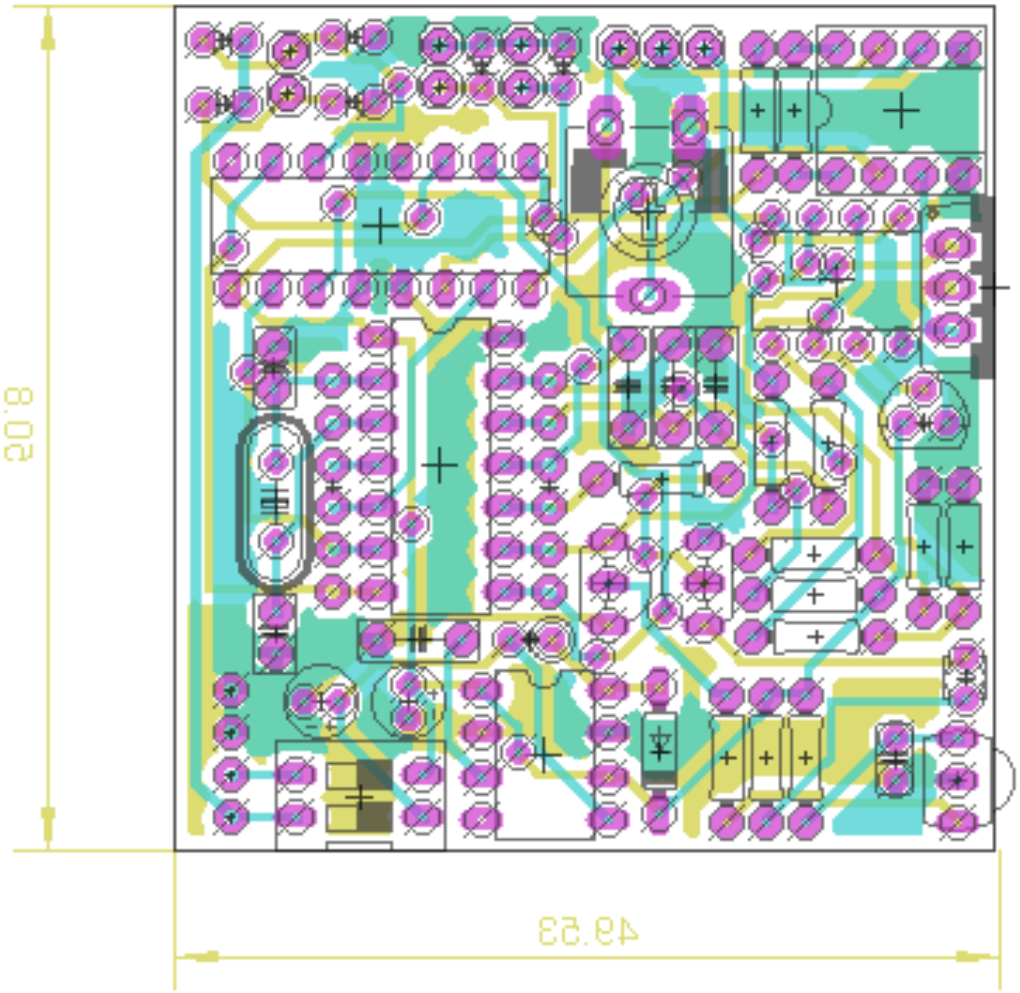


Fig. 7.2 PCB

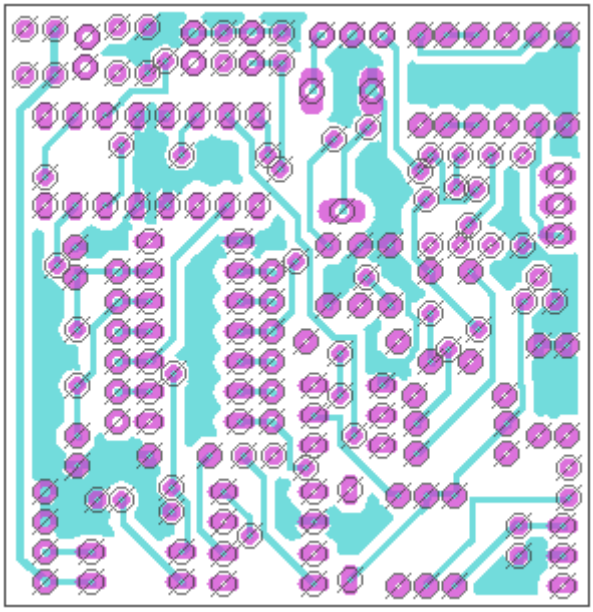


Fig. 7.3 PCB Cara Superior

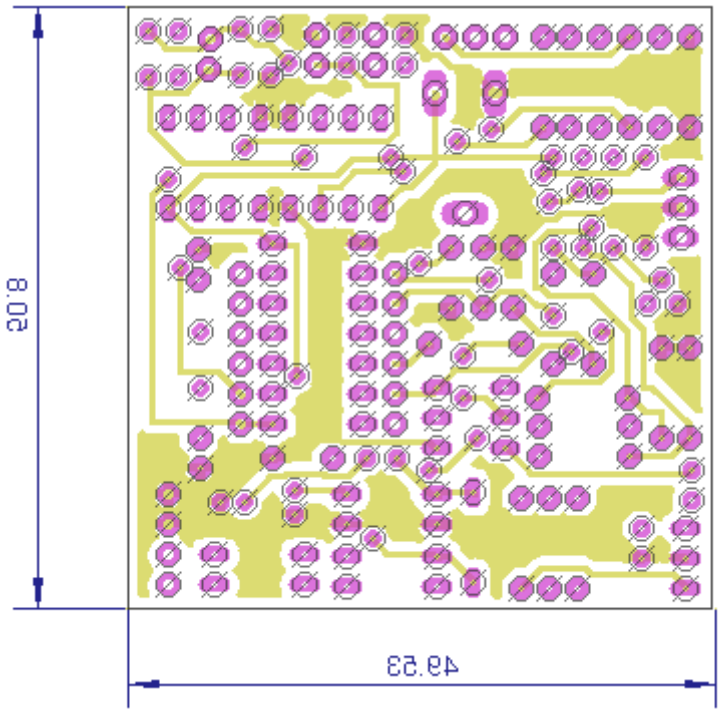


Fig. 7.4 PCB Cara Inferior

7.2 Imágenes del sistema completo

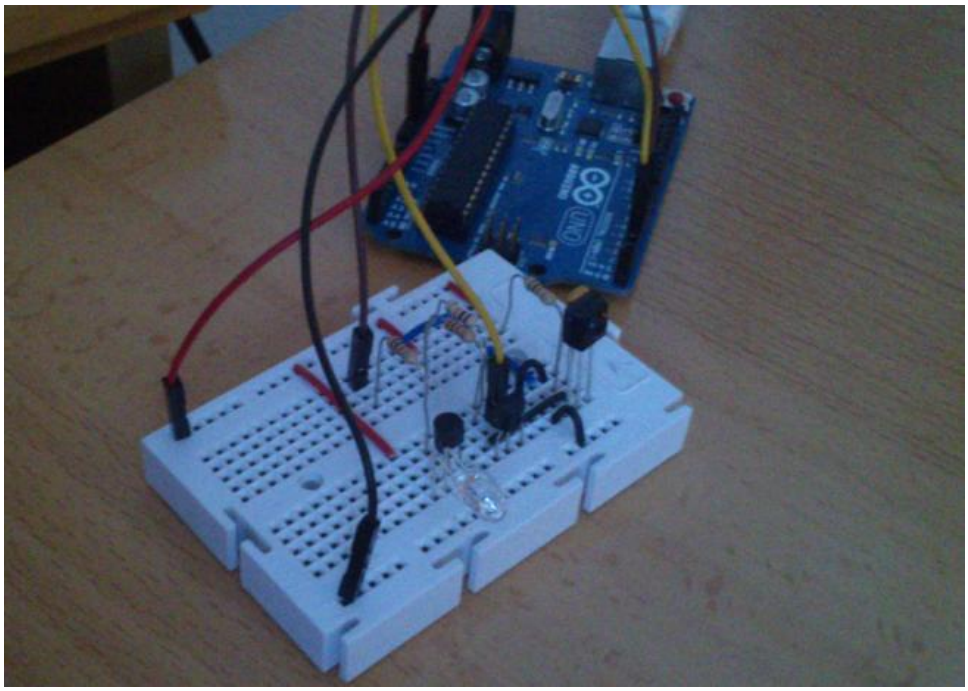


Fig. 7.5 Imagen de la estación base, Arduino UNO

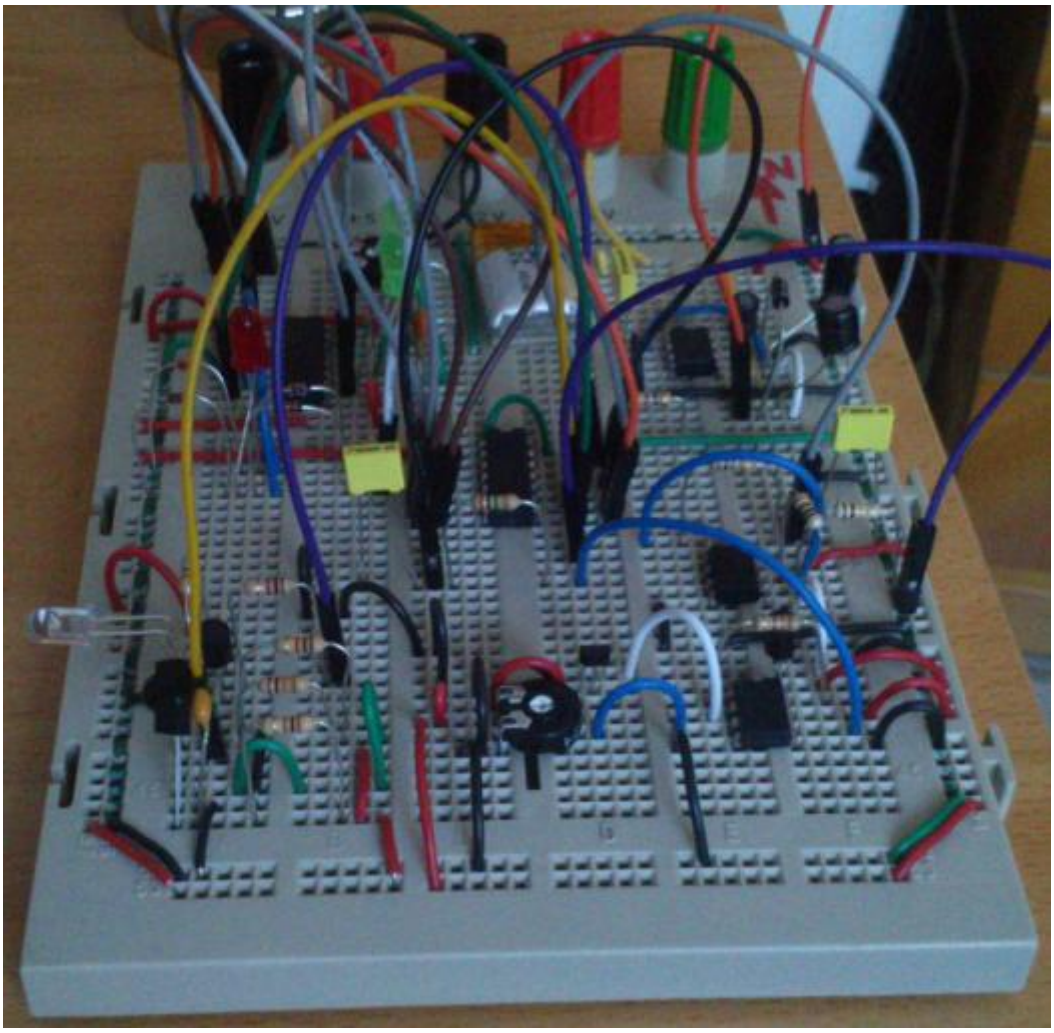


Fig. 7.6 Imagen de la estación remota, ATtiny84.

ANEXO 8. DATASHEETS

8.1 LM35DZ

Datasheet completo: www.ti.com/lit/ds/symlink/lm35.pdf

National Semiconductor

November 2000

LM35
Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications

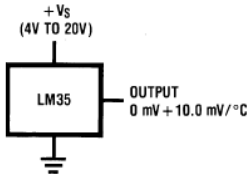
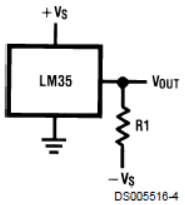


FIGURE 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)



Choose $R_1 = -V_S/50$ µA
V_{OUT} = +1,500 mV at +150°C
= +250 mV at +25°C
= −550 mV at −55°C

FIGURE 2. Full-Range Centigrade Temperature Sensor

© 2000 National Semiconductor Corporation DS005516 www.national.com

Absolute Maximum Ratings (Note 10)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage	+35V to −0.2V
Output Voltage	+6V to −1.0V
Output Current	10 mA
Storage Temp.;	
TO-46 Package,	−60°C to +180°C
TO-92 Package,	−60°C to +150°C
SO-8 Package,	−65°C to +150°C
TO-220 Package,	−65°C to +150°C
Lead Temp.:	
TO-46 Package, (Soldering, 10 seconds)	300°C
TO-92 and TO-220 Package, (Soldering, 10 seconds)	260°C
SO Package (Note 12)	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
ESD Susceptibility (Note 11)	2500V
Specified Operating Temperature Range: T_{MIN} to T_{MAX} (Note 2)	
LM35, LM35A	−55°C to +150°C
LM35C, LM35CA	−40°C to +110°C
LM35D	0°C to +100°C

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35A			LM35CA			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy	T_A=+25°C	±0.2	±0.5		±0.2	±0.5		°C
(Note 7)	T_A=−10°C	±0.3			±0.3		±1.0	°C
	T_A=T_{MAX}	±0.4	±1.0		±0.4	±1.0		°C
	T_A=T_{MIN}	±0.4	±1.0		±0.4		±1.5	°C
Nonlinearity (Note 8)	$T_{MIN} \le T_A \le T_{MAX}$	**±0.18**		**±0.35**	**±0.15**		**±0.3**	°C
Sensor Gain (Average Slope)	$T_{MIN} \le T_A \le T_{MAX}$	**+10.0**	**+9.9, +10.1**		**+10.0**		**+9.9, +10.1**	mV/°C
Load Regulation (Note 3) $0 \le I_L \le 1$ mA	T_A=+25°C	±0.4	±1.0		±0.4	±1.0		mV/mA
	$T_{MIN} \le T_A \le T_{MAX}$	**±0.5**		**±3.0**	**±0.5**		**±3.0**	mV/mA
Line Regulation (Note 3)	T_A=+25°C	±0.01	±0.05		±0.01	±0.05		mV/V
	$4V \le V_S \le 30V$	**±0.02**		**±0.1**	**±0.02**		**±0.1**	mV/V
Quiescent Current (Note 9)	V_S=+5V, +25°C	56	67		56	67		µA
	V_S=+5V	**105**		**131**	**91**		**114**	µA
	V_S=+30V, +25°C	56.2	68		56.2	68		µA
	V_S=+30V	**105.5**		**133**	**91.5**		**116**	µA
Change of Quiescent Current (Note 3)	$4V \le V_S \le 30V$, +25°C	0.2	1.0		0.2	1.0		µA
	$4V \le V_S \le 30V$	**0.5**		**2.0**	**0.5**		**2.0**	µA
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.5**	**+0.39**		**+0.5**	µA/°C
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	°C
Long Term Stability	T_J=T_{MAX}, for 1000 hours	±0.08			±0.08			°C

8.2 Sensor efecto HALL A1301

Datasheet completo:
http://pdf1.alldatasheet.es/datasheet-pdf/view/120794/ALLEGRO/A1301.html

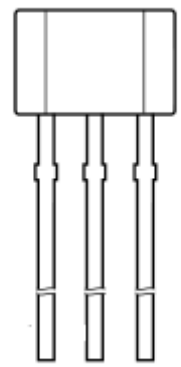
A1301 and A1302 — ***Continuous-Time Ratiometric Linear Hall Effect Sensor ICs***

DEVICE CHARACTERISTICS over operating temperature range, T_A, and V_{CC} = 5 V, unless otherwise noted

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Electrical Characteristics						
Supply Voltage	V_{CC}	Running, T_J < 165°C	4.5	–	6	V
Supply Current	I_{CC}	Output open	–	–	11	mA
Output Voltage	$V_{OUT(High)}$	I_{SOURCE} = –1 mA, Sens = nominal	4.65	4.7	–	V
	$V_{OUT(Low)}$	I_{SINK} = 1 mA, Sens = nominal	–	0.2	0.25	V
Output Bandwidth	BW		–	20	–	kHz
Power-On Time	t_{PO}	$V_{CC(min)}$ to 0.95 V_{OUT}; B = ±1400 G; Slew rate = 4.5 V/µs to 4.5 V/100 ns	–	3	5	µs
Output Resistance	R_{OUT}	I_{SINK} ≤ 1 mA, I_{SOURCE} ≥ –1 mA	–	2	5	Ω
Wide Band Output Noise, rms	V_{OUTN}	External output low pass filter ≤ 10 kHz; Sens = nominal	–	150	–	µV
Ratiometry						
Quiescent Output Voltage Error with respect to ΔV_{CC}[1]	$\Delta V_{OUTQ(V)}$	T_A = 25°C	–	–	±3.0	%
Magnetic Sensitivity Error with respect to ΔV_{CC}[2]	$\Delta Sens_{(V)}$	T_A = 25°C	–	–	±3.0	%
Output						
Linearity	Lin	T_A = 25°C	–	–	±2.5	%
Symmetry	Sym	T_A = 25°C	–	–	±3.0	%
Magnetic Characteristics						
Quiescent Output Voltage	V_{OUTQ}	B = 0 G; T_A = 25°C	2.4	2.5	2.6	V
Quiescent Output Voltage over Operating Temperature Range	$V_{OUTQ(\Delta T_A)}$	B = 0 G	2.2	–	2.8	V
Magnetic Sensitivity	Sens	A1301; T_A = 25°C	2.0	2.5	3.0	mV/G
		A1302; T_A = 25°C	1.0	1.3	1.6	mV/G
Magnetic Sensitivity over Operating Temperature Range	$Sens_{(\Delta T_A)}$	A1301	1.8	–	3.2	mV/G
		A1302	0.85	–	1.75	mV/G

[1]Refer to equation (4) in Ratiometric section on page 4.
[2]Refer to equation (5) in Ratiometric section on page 4.

Package UA



VCC GND VOUT

BIBLIOGRAFÍA Y REFERENCIAS

[1] Excelente manual sobre microcontroladores AVR:
http://www.cursomicros.com/avr/arquitectura/fuses-de-los-avr.html

[2] Fig. 1.2 Fuses para utilizar un cristal externo de 16 MHz:
http://www.engbedded.com/fusecalc/

[3] Fig. 1.3 Conector de salida del programador AVR:
http://wiki.edwindertien.nl/doku.php?id=modules:sdadapter

[4] Fig. 1.4 Esquema para programar fuses del ATtiny:
Esquema realizado con fritzing, http://fritzing.org/

[5] Fig. 1.5 y 1.6 Pantalla del programa PROISP v 1.72
Programa PROGISP, http://getavr.wordpress.com/download/

[6] Fig. 2.2 Simbología de un transistor NPN
http://commons.wikimedia.org/wiki/File:BJT_symbol_NPN.svg

[7] Fig. 3.12 Principales características de un diodo emisor OPE5685
http://pdf1.alldatasheet.es/datasheet-pdf/view/95912/ETC/OPE5685.html

[8] Fig. 3.15 Principales características de un receptor TSOP4P38.
http://test-market.interactive-ic.com/static/content/pdfs/195/6256398.pdf